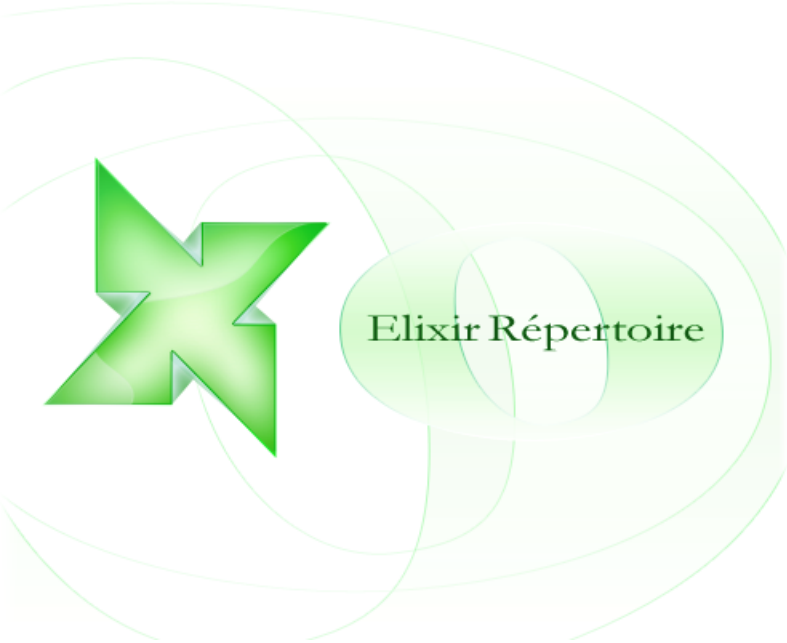
Elixir Data Designer User Manual

Release 8.0



Elixir **Technology Pte Ltd**

Elixir Data Designer User Manual: Release 8.0

Elixir Technology Pte Ltd

Published 2010
Copyright © 2004-2010 Elixir Technology Pte Ltd

All rights reserved.

Solaris, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. Microsoft and Windows are trademarks of Microsoft Corporation.

Table of Contents

1. About Elixir Data Designer 1
    Introduction 1
        Features of Elixir Data Designer 1
        Datasource Security 2
2. The Elixir Data Designer Workspace 3
    Overview 3
    Case Study 4
3. JDBC DataSource 7
    JDBC Drivers 7
    ODBC Connectivity 8
    JDBC DataSource Wizard 9
        JDBC DataSource Details 9
        SQL 10
        Infer Schema 13
        JDBC Options 13
    Connection Pools 14
    Working with a JDBC DataSource 16
        Using the JDBC/ODBC bridge driver 17
        Using a Callable Statement 17
        Using JNDI Connectivity 19
4. Composite DataSource 21
    Overview 21
    Adding a Composite DataSource 21
    Standard Diagram Operations 22
        Moving the processors 22
        Flow Connection Nodes 22
    Data Processors 22
    DataSource Processor 23
    Flow 24
    Join Processor 24
        Working with Joins 26
    Sort Processor 29
        Group On Options 31
        Working With Sort processors 34
    Derivative Processor 39
        Working with the Derivative processor 41
    Filter Processor 45
        Combining Filters 46
        Working with Filters 47
    Concat Processor 49
        Working with Concat Processor 49
    Parameter Processor 50
        Working with Parameter processor 51
    SubFlow Processor 53
    Processor 54
        Cleansing 54
        General 55
    DataDrop Processor 58
    Note 59
    Cube Processor 59
        Working with Cube Processor 65
    DataStore Processor 67
        CSV File 68
        Connection Pool 69
        Custom Java DataStore 69

Excel File 70
JDBC 71
None 72
Text File 72
XML File 72
Composite JavaScript 73
Case Study 75
5. Excel DataSource 80
Overview 80
Excel DataSource Wizard 80
Working with Excel DataSource 81
Adding Data in a Single Worksheet 81
Adding Data in Multiple Worksheets 83
Passing Dynamic Parameters 86
6. Text DataSource 90
Overview 90
Text DataSource Wizard 90
Working with Text DataSource 94
Using Separator Characters 94
Defining a URL with a Dynamic Parameter 95
Using Fixed Width Columns 96
Using Regular Expressions 97
Using Start and Stop Expressions 99
7. XML DataSource 100
Overview 100
XML DataSource Wizard 101
Use of Merge 103
Subtree Optimization 104
HTML 104
EDIFACT 104
Search in XPath Builder 105
Working with XML DataSources 105
8. Properties DataSource 108
Overview 108
Properties DataSource Wizard 108
Working with Properties DataSources 109
Testing the Data Flow 109
Passing Parameters to the Flow 113
9. Reference DataSource 116
Overview 116
Reference DataSource Wizard 116
Working with Reference DataSource 117
Wrapping an Excel DataSource 117
Wrapping a Composite DataSource 120
10. Object DataSource 123
Overview 123
The Classpath 123
Object DataSource Wizard 123
Guided JavaScript Builder 124
Manual Creation 125
Object DataSource API 127
PushContext 127
DataListener 127
DataGroup 128
DataRecord 128
Working with Object DataSources 128
11. LDAP DataSource 131
Overview 131

LDAP DataSource Wizard 131
Working with LDAP DataSource 134
12. Filesystem DataSource 137
Overview 137
Filesystem DataSource Wizard 137
Filesystem Schema 138
FileSystem DataSource in DBFS FileSystem In Remote Designer 138
13. Tabular DataSource 140
Overview 140
Tabular DataSource Wizard 140
14. Random DataSource 142
Overview 142
Random DataSource Wizard 142
Expression 143
Literal 144
Script 144
15. ARFF DataSource 145
Overview 145
ARFF DataSource Wizard 145
16. Cache Datasource 147
Overview 147
Cache Datasource Wizard 147
17. DBF Datasource 149
Overview 149
DBF Datasource Wizard 149
A. Dynamic Parameters 151
Dynamic Parameters 151
Dynamic Parameter Elements 151
Dynamic Parameter Names 151
Dynamic Parameter Types 151
Ordering Dynamic Parameters 152
Dynamic Parameters with a Nested DataSource 153
Example Declaration of Dynamic Parameters. 154
B. Samples 156
Availability 156

List of Figures

1.1. Set Security Options 2
2.1. Basic Data Workspace 3
2.2. Composite Data Workspace 4
2.3. New DataSource 5
2.4. Define Text DataSource - Page One 5
2.5. Define Text DataSource - Page Two 6
3.1. Create New Data Source 8
3.2. Select Database 9
3.3. JDBC DataSource Wizard 9
3.4. Define JDBC DataSource 11
3.5. Elixir Query Builder 11
3.6. Callable Tab 12
3.7. Define DataSource Schema 13
3.8. Define DataSource Options 14
3.9. Connection Pool Wizard 14
3.10. Right-click on Connection Pool 16
3.11. Datasource Wizard 16
3.12. JNDI Values 18
4.1. Composite DataSource 21
4.2. Result Wizard 22
4.3. DataSource Wizard 23
4.4. Sample Flow 24
4.5. Join Wizard 25
4.6. Sample Join Flow 26
4.7. Join Wizard 27
4.8. Inner Join Result 27
4.9. Outer Join Result 28
4.10. Cross Join Result 29
4.11. Sort Wizard 30
4.12. Add Sort Item 30
4.13. Extract Options 31
4.14. Sample Sort Flow 34
4.15. Sort Dialog 36
4.16. Group on - Each Result 36
4.17. Group on - All Result 37
4.18. Group on - Range Result 37
4.19. Group on - Range by Country Result 38
4.20. Completed Sort Dialog 39
4.21. Derivative Wizard 40
4.22. Add Column Dialog 40
4.23. Sample Derivative Flow 41
4.24. Completed Add Column Dialog 42
4.25. Derived Result 42
4.26. Completed Add Column Screen 44
4.27. Date Manipulations Result 44
4.28. Filter Wizard 46
4.29. Sample Filter Flow 48
4.30. Filter Result 48
4.31. Concat Wizard 49
4.32. Sample Concat Flow 50
4.33. Parameter Wizard 51
4.34. Completed Properties DataSource 52
4.35. Sample Parameter Flow 52
4.36. Parameter Result 53
4.37. SubFlow Sample 53

4.38. Input Wizard 54
4.39. Remove Duplicates Processor 55
4.40. Invert Data Processor 56
4.41. Javascript Processor 57
4.42. Map Field Wizard 57
4.43. SQL Processor Wizard 58
4.44. Sample DataDrop Flow 59
4.45. Sample DataDrop Sub Flow 59
4.46. Cube Wizard 60
4.47. Add Hierarchy 61
4.48. Cube Axes Screen 61
4.49. Cube Measures Screen 62
4.50. Add Measure Dialog 62
4.51. Cube - Show Row Grand Totals 63
4.52. Cube - Show Column Grand Totals 64
4.53. Cube - Row Totals After Details 64
4.54. Cube - Column Totals After Details 65
4.55. Cube Axes - Edit Level 65
4.56. Sample Cube Flow 66
4.57. Infer Schema Screen 66
4.58. Cube Result 67
4.59. Sample Datastore Flow 68
4.60. CVS Output 68
4.61. DataStore Wizard - Connection Pool 69
4.62. Custom Java Option Screen 70
4.63. Excel Output 71
4.64. XML Output 73
4.65. Case Study Composite Diagram 76
4.66. View Cube Output 78
5.1. Excel DataSource Wizard 80
5.2. Excel DataSource Wizard 81
5.3. EmpInfo.xls 82
5.4. Define Name 82
5.5. Completed DataSource Wizard 83
5.6. Result 83
5.7. EmpDetail.xls 84
5.8. Define Name 84
5.9. Completed DataSource Wizard 85
5.10. Result 86
5.11. EmpDet.xls 86
5.12. Define Name 87
5.13. Completed DataSource Wizard 88
5.14. Dynamic Parameters 89
5.15. Emp1_Small Result 89
5.16. Emp1_All Result 89
6.1. Define Text DataSource 90
6.2. Separator Type Properties 91
6.3. Fixed Width Type Properties 92
6.4. Regular Expression Type Properties 93
6.5. Regular Expression Designer 93
6.6. Sample Text DataSource 95
6.7. Dynamic Parameters 96
6.8. Fixed Width Sample 97
6.9. Log file 97
6.10. Regular Expression Designer 98
7.1. An XML Tree 100
7.2. Define XML DataSource 101
7.3. Choosing the XPath Builder 102

7.4. Merge Space 103
7.5. Merge Group 104
7.6. XPath Builder 105
7.7. XPaths Completed 106
7.8. XML DataSource Schema 107
7.9. DataSource Results 107
8.1. Properties DataSource Wizard 108
8.2. Add Column 109
8.3. Text DataSource Parameters 110
8.4. Properties DataSource 111
8.5. Derivative Wizard 112
8.6. Test Output 112
8.7. Full Output 113
8.8. Prop_Emp1 113
8.9. Filter Wizard 114
8.10. Output 114
9.1. Reference DataSource Wizard 116
9.2. Reference DataSource Wizard Page Two 117
9.3. Empdata.xls 117
9.4. Define Name 118
9.5. Excel DataSource Sample 119
9.6. Reference a DataSource 119
9.7. Sample Output 120
9.8. Filter Condition 121
9.9. Sample Reference 121
10.1. Object DataSource Wizard 124
10.2. Guided JavaScript Builder 125
10.3. JavaScript Editor 126
10.4. Sample Data Iterator 129
10.5. Sample Data Class 129
11.1. LDAP DataSource Wizard 132
11.2. Security Parameters 133
11.3. LDAP DataSource Schema 134
11.4. LDAP Datasource Wizard 135
11.5. Completed Datasource Wizard 135
11.6. Organization Result 136
12.1. Filesystem DataSource Wizard 137
12.2. File Attributes 139
13.1. Tabular DataSource Wizard 140
13.2. Tabular DataSource Wizard Page Two 141
14.1. Random DataSource Wizard 142
14.2. Edit Column 143
15.1. ARFF DataSource Wizard 145
15.2. Define ARFF Schema 146
16.1. Cache DataSource Wizard 149
16.2. Cache DataSource Parameters 150
17.1. DBF DataSource Wizard 149
17.2. Enter DBF Schema 150

List of Tables

4.1. Filter Criteria 47

Chapter 1
About Elixir Data Designer

Introduction

Elixir Data Designer is a tool used for creation and manipulation of data. It allows direct interaction with the data flow through a diagrammatic interface.

Users of IT systems face common problems:

- Growing volume of data - IT and business users are required to handle large volumes of data every day. Manual processing is both time-consuming and error-prone, leading to inefficient operations and an accumulation of mistakes.
- Proliferation of data - Business users work with different types of databases or applications. This leads to more complexity in data retrieval and manipulation.

The solution to these problems is to personalize the data sources, so that it becomes easier to manage them. Elixir Data Designer is a powerful tool that can perform extraction, aggregation, transformation and loading of multiple data sources. Further, since the designer provides a visual method of interaction with these data sources, a large amount of time can be saved while developing data manipulation strategies.

Elixir Data Designer provides on-demand data for business users. As this software is directly deployable from the web, it can be easily accessed by business users to satisfy their everyday needs. The tool also provides a flexible scripting mechanism, allowing an IT support team to build custom solutions that satisfy complex requirements.

Through direct manipulation and inspection of data sources and flows, a coherent view of the overall data architecture can be gained. Time spent developing custom codes and programs can be avoided by using the tool to formulate the data processors and intercept the flow at any stage to ensure correctness. Finally, the output can be generated into a variety of different data formats, including Microsoft Excel, relational database tables and Elixir Report Designer.

Features of Elixir Data Designer

Elixir Data Designer includes:

Multiple Data Source Support: This tool is capable of manipulating and merging data from many data sources, including custom data sources. Standard data source support includes JDBC, XML, Microsoft Excel, and a variety of text formats.

Built-in Filtering: When only a particular subset of data is required, it can be extracted using the Filter processor available in the Composite Designer. More complex filtering can be performed with the help of scripts.

Data Derivation: By using the Derivative processor, a new field can be derived by performing operations on the existing columns in the data source.

Multi-level Sorting: The data can be sorted and grouped using the Sort processor.

Data Aggregation: Using the Composite Join processor, any two data sources can be joined together. A comprehensive set of join operations is supported, including inner join, outer join and cross join.

Multi-Dimensional Data Transformation: The OLAP Cube control can be used to perform complex data operations with hierarchies, dimensions and multiple measures using standard operations like max, min, count, etc.

Caching: The Cache processor is used to cache the data for a specified amount of time so that the user can avoid unnecessary repeated data retrieval from the original source. The related staging feature stores the data temporarily when all the processes are completed. This helps to flush/optimise the memory.

Data Output and Loading: The different data store types available in Elixir Data Designer include XML File, JDBC, Excel File and CSV File. In addition, through the Custom Java DataStore, user-defined stores can be supported.

Datasource Security

At the last page of the DataSource Wizard, the user can set security options, as seen in Figure 1.1, “Set Security Options”.

Figure 1.1. Set Security Options

- *Read-Only*: When selected and saved, the next time a user opens this datasource, the user will not be able to edit any details of the datasource like name, description and column names.
- *Hide Details*: When this option is selected and saved, the next time when this datasource is opened, the user will only be able to see the name and description of the datasource.
- *Encrypted*: This option is to be used with either `Read-Only` option or `Hide Details` option or both. Checking the checkbox will prompt the user to enter a password, then re-enter to confirm the password. (Both passwords must be the same) After this is done, if another user would like to edit any selections, he will need to enter the password.

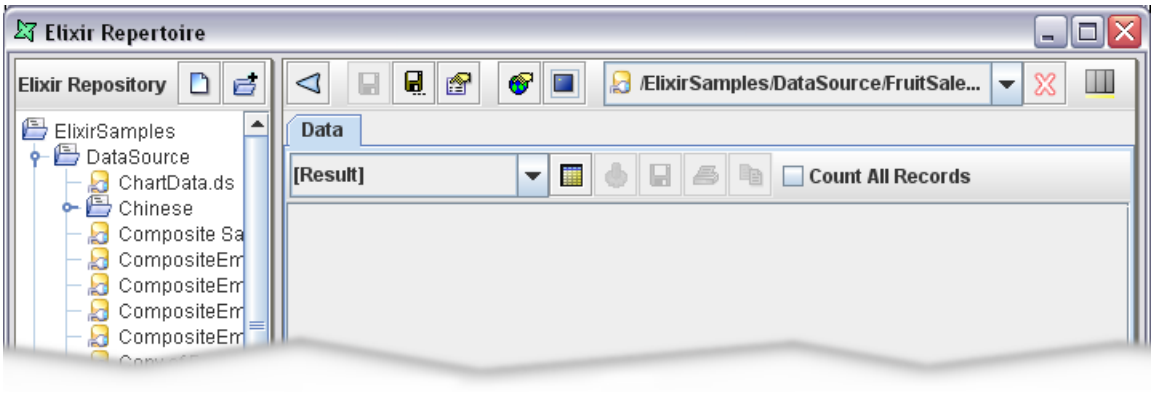
Chapter 2
The Elixir Data Designer Workspace

Overview

The Elixir Repertoire application, consists of three parts - the Elixir Repository, Action Bar and Workspace. These are discussed in the Elixir Repertoire manual. The data designer contributes new views to the Workspace.

The workspace of the basic data sources like Excel DataSource, Text DataSource, etc. consists of a Data tab panel as shown in Figure 2.1, “Basic Data Workspace”.

Figure 2.1. Basic Data Workspace



The Data tab provides the following functions:

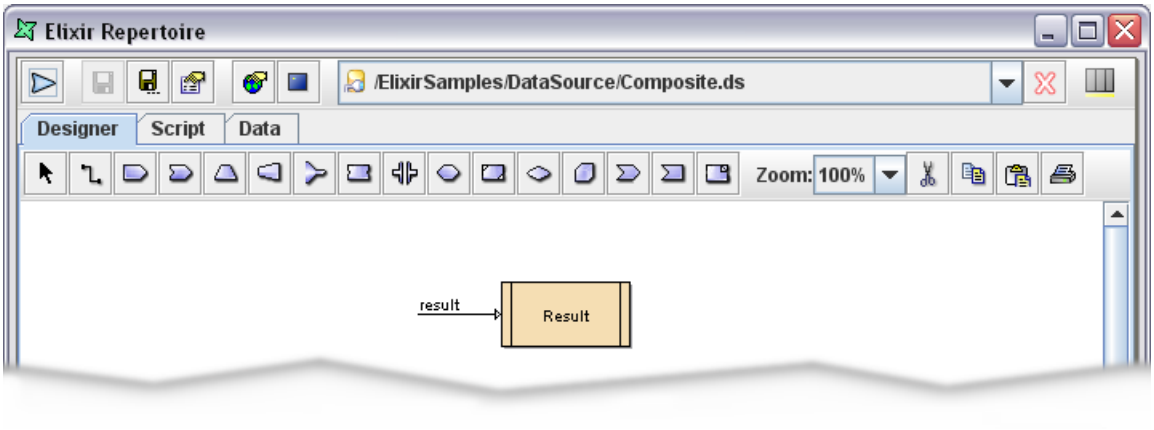
- *Load Data*: Initiates the data flow to process the data source records and render the output as a table within this data tab.
- *Save Data*: Initially, the Save Data icon is disabled. It is only activated after data has been loaded. Using this option, data can be saved in different file formats.
- *Print Data*: Initially, the Print Data icon is disabled. Once data has been loaded the option is enabled. Using this option, the data table can be printed out.
- *Copy*: When a data table has been loaded, rows can be selected for copying to the clipboard. The set of data copied can be pasted into an editor and saved as an XML file.
- *Count All Records*: This option is used to display the total number of records present in the data source. By default the Data Window displays up to a maximum of 500 records. On selecting the "Count All Records" check box and clicking the Load Data icon the total number of records available in the data source is displayed on top of the Data Window. Turning this option on will slow the response for huge data sources as the value can only be obtained after all processing has finished.
- *Generate Data*: When clicked, data in the datasource will be generated and displayed.

The Composite DataSource workspace consists of three tabbed panels: Designer, Script and Data.

Designer Tab

The Designer tab is shown in Figure 2.2, “Composite Data Workspace”. In order to show all the icons within the width of this document, the repository tree on the left has been collapsed. There are various Data Operation constructs present in the toolbar of the Designer tab. The Designer window is the main window where the data flow is designed with the help of the Data Operation constructs.

Figure 2.2. Composite Data Workspace



Script Tab

The Script tab allows the creation of JavaScript functions to enhance the processing power of the Elixir Data engine.

Data Tab

The third window is the Data tab, which is identical to the tab shown in Figure 2.1, “Basic Data Workspace” for basic data sources.

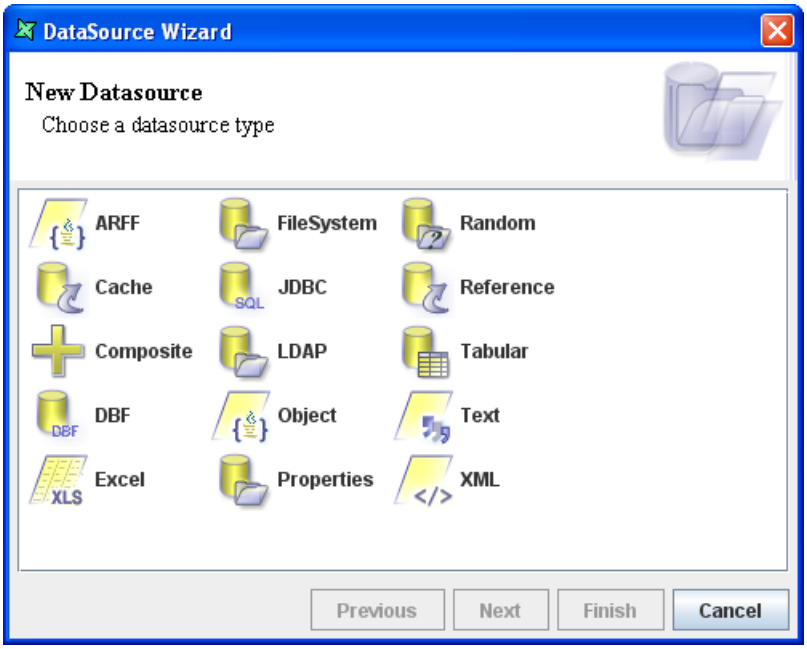
Case Study

The details of employees working in a company are maintained in a file. We want to view, count and print out the records.

> ## Note
>
> See Appendix B, *Samples* for details of the sample files that support the case studies in this manual.

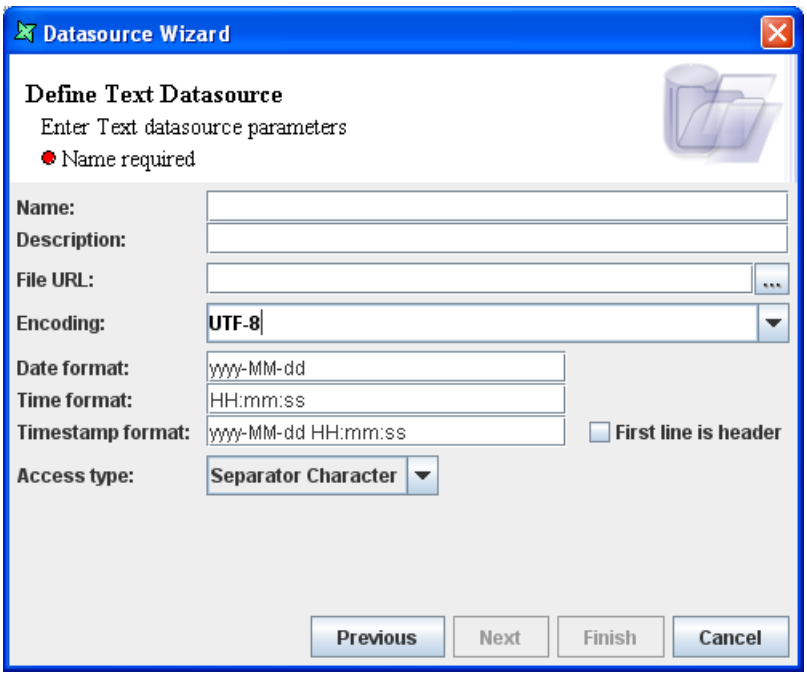
1. Choose a file system or folder and select Add -> DataSource from the popup menu, the DataSource Wizard appears as shown in Figure 2.3, “New DataSource”. Select the Text DataSource from the DataSource types and click the Next button.

Figure 2.3. New DataSource



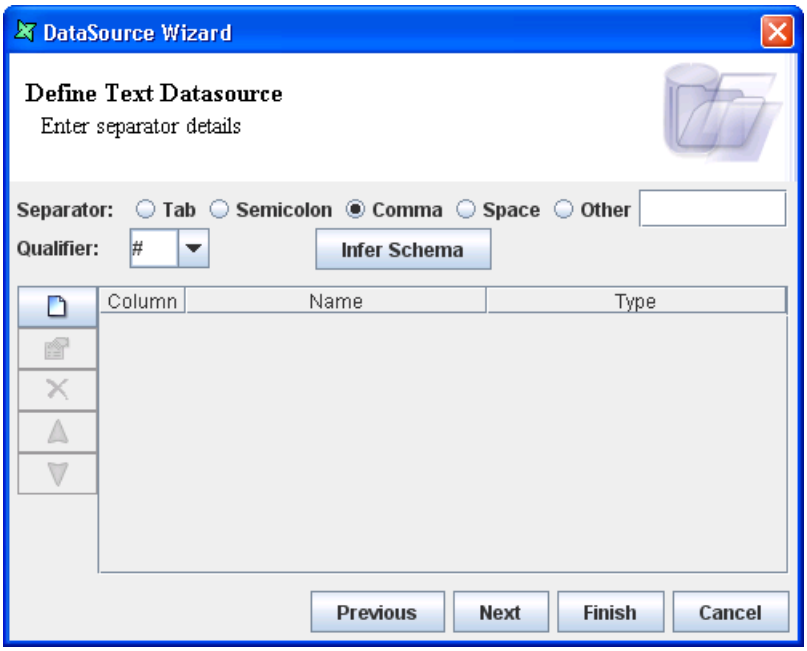
2. The Define Text DataSource screen appears as shown in Figure 2.4, “Define Text DataSource - Page One”. On this page, enter `Employee` in the Name field. Click the browse button to the right of the File URL text box and select the path to the "Employee.txt" file from the Open File Dialog window. Alternatively, if Employee.txt is in the repository, enter repository: followed by the path to the file. In all of the samples (see Appendix B, *Samples*) the repository path is used and in this case it is repository:/DataDesignerUserManual/data/Employee.txt. Tick the "First line is header" check box. Click the Next button.

Figure 2.4. Define Text DataSource - Page One



3. The screen as shown in Figure 2.5, “Define Text DataSource - Page Two” appears. In this screen click the Infer Schema button. You should see the field types from the employee text file appear. Click the Finish button. The Employee.ds file appears in the repository under the chosen file system.

Figure 2.5. Define Text DataSource - Page Two



4. Now to load the data of the Employee data source the manager has to click the Load Data icon in the Data window. To print the data, just click the Print Data icon.

5. Only the first five hundred records are processed in this table. The record count will automatically stop at this limit. If you have a larger number of records, you can select the "Count All Records" check box in the data window before the Load Data icon. Now the total number of records present in the data source is displayed in the toolbar of the Data Window. In this example there are only ten records, so "Count All Records" won't affect the result.

6. Finally click the close icon on the right of the toolbar to close the view.

Note

Please refer to Appendix B, *Samples* for the sample files used in this example.

Chapter 3
JDBC DataSource

Java Database Connectivity (JDBC) is a standard SQL database access interface, providing uniform access to a wide range of relational databases. JDBC also provides a common base on which higher level tools and interfaces can be built. Elixir Data Designer supports reading of data using JDBC and provides an SQL query builder.

JDBC Drivers

There are four categories of JDBC drivers.

1. *JDBC-ODBC bridge plus ODBC driver* - This driver provides JDBC API access via one or more ODBC drivers. ODBC native code and native database client code may be required on each machine that uses this type of driver.

2. *Native API partly-Java driver* - A native API partly Java technology-enabled driver converts JDBC calls into calls to the client API for Oracle, Sybase, Informix, DB2, or other DBMS. This driver is similar to the ODBC bridge driver as it requires some binary code to be loaded on each client machine.

3. *JDBC-Net pure Java driver* - This type of driver translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect its clients to different databases. The specific protocol used depends on the vendor.

4. *Native protocol pure Java driver* - This driver converts JDBC technology calls into the network protocol used by a specific DBMS directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for intranet access.

To illustrate the configuration process for a driver, here are the steps on how to use a MySQL DataSource. The latest MySQL driver can be downloaded from http://www.mysql.com/. The mysql jar file must be copied to the Elixir Repertoire ext folder. Similarly, for an Oracle DataSource, the latest driver must be copied to the ext folder which can be downloaded from http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc9201.html [http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc9201.html]. For a Postgres driver, check here: http://jdbc.postgresql.org/.

In each case, regardless of the DBMS vendor, the corresponding driver files are copied to the ext folder. Elixir Repertoire must be restarted to load any new drivers added.

Note

If you are using Elixir Repertoire Remote, then the database drivers must be placed in the server ext folder instead. This is because data operations are performed on the server. The only exception is the Query Builder functionality which requires access to the database schema information. See the note in the Query Builder section below describing how to configure the Remote tool to support this.

ODBC Connectivity

Before using the JDBC-ODBC bridge to connect to data sources, the ODBC connectivity must be established.

The Open Database Connectivity (ODBC) interface is a Microsoft standard for accessing different database systems from Windows. The ODBC interface permits maximum interoperability - an application can access data in diverse DBMSs through a single interface. Furthermore, the application will be independent of the DBMS from which it accesses data. Users of the application can add software drivers, which interface between Elixir Data Designer and a specific DBMS. Note that Sun's JDBC-ODBC interface is considered a reference implementation, hence it is not tuned for performance.

The Microsoft ODBC Data Source Administrator manages database drivers and data sources.

To establish an ODBC Connection you need to:

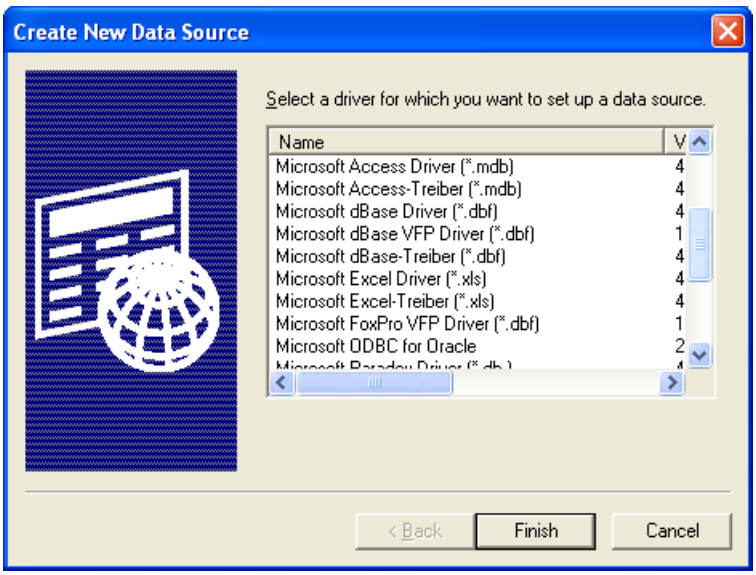
1. Click Start -> Settings -> Control Panel. Different versions of Windows provide different graphical interfaces to the ODBC administration tools.

   - On a Windows 95/98/NT PC double-click on the ODBC Data Sources (32bit) icon to open the ODBC Data Source Administrator.

   - On a Windows 2000/2003/XP/Vista/2008 PC double-click on the Administration Tools icon and then double-click on the ODBC Data Sources icon to open the OCBC Data Source Administrator.

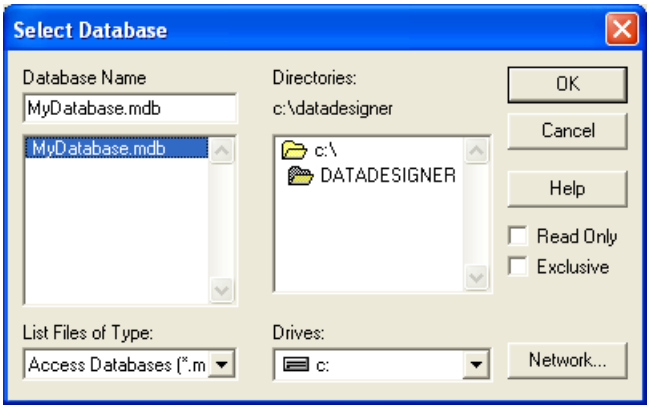
2. Clicking the Add button shows the dialog in Figure 3.1, “Create New Data Source”. The appropriate driver can be selected from the list of User Data Sources. Click Finish to continue.

Figure 3.1. Create New Data Source



3. The "ODBC Microsoft Access Setup" dialog box pops up. Here the Data Source name is entered in the text box. Next click the Select button and select the path to the database from the dialog box as shown in Figure 3.2, “Select Database”.

Figure 3.2. Select Database

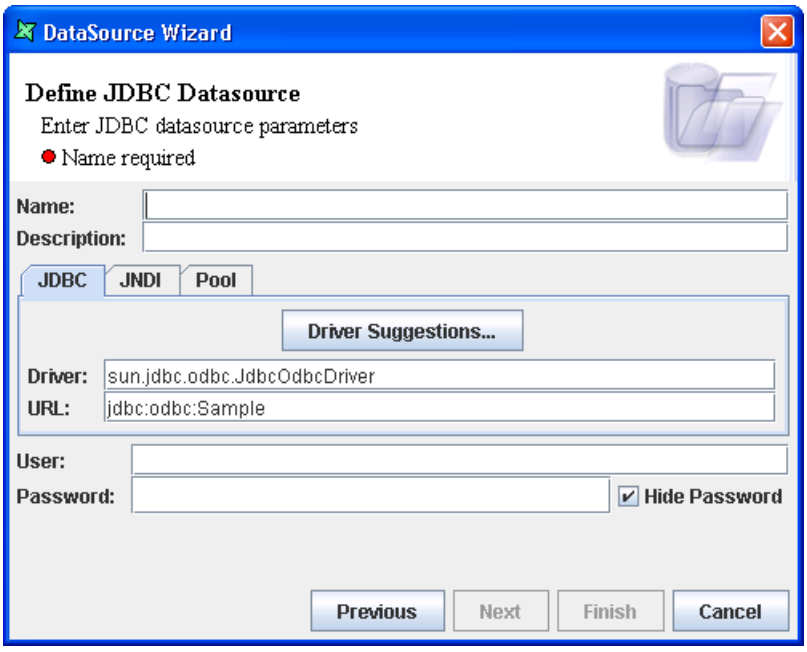


4. The selected database should now have been added to the list of existing databases. Click the OK button in the "ODBC Data Source Administrator" dialog to complete this task.

JDBC DataSource Wizard

The JDBC DataSource Wizard allows a new JDBC DataSource to be created. The wizard is shown in Figure 3.3, “JDBC DataSource Wizard”

Figure 3.3. JDBC DataSource Wizard



JDBC DataSource Details

"Define JDBC DataSource" is the first screen of the DataSource wizard. In this screen, JDBC DataSource parameters can be entered.

- *Name*: A unique name to identify the DataSource.
- *Description*: Extra description for the data source.

- *JDBC connection*: Three methods of connection are available: JDBC, JNDI or Pool.

  a) JDBC

  Driver Suggestions: The type of the JDBC driver can be selected from the Driver Suggestions combo box. A green or red symbol next to each driver indicate whether the driver is available for use. If the specific driver jar file is copied to the ext folder before Elixir Repertoire is launched, a green symbol will appear next to that driver indicating that the driver has been loaded and is available for use. On the Remote Designer, it works slightly differently. Only JDBC drivers that are available (which have green symbols) will be listed.

  Driver: When the type of the Driver is selected from the Driver Suggestions combo box the default Driver is automatically entered in the Driver Text box. The Driver class name in the text box can be altered in case your DBMS vendor modifies the class name.

  URL: When the type of the Driver is selected from the Driver Suggestions combo box the default URL is automatically entered in the URL Text box. The parameters in the URL text box can be altered according to your JDBC vendor's requirements.

  b) JNDI

  Context Factory: The context factory accepts information about how to create a context, such as a reference, and returns the instance of the context.

  Provider URL: Provides the URL of the resource to bind to.

  Resource Name: This identifies the resource name, that binds to the data source.

  c) Pool

  Connection Pool: The connection pool from the Repository that will provide the connection to the database. See the section called “Connection Pools”, later in this chapter, for more details.

- *User*: The user name is entered if required.

- *Password*: The password is entered if required.

SQL

When the data source parameters have been entered, click Next to see the screen as shown in Figure 3.4, “Define JDBC DataSource”. In this window there are two tabbed panes namely the SQL tab and the Callable tab. An SQL Statement can be entered in the SQL editor. Stored procedures can be defined in the Callable tab. The Query Builder button can also be used to build a query using visual tools.

When `Time SQL` button is clicked, there will be a pop-up window displaying the number of records retrieved and the time taken to do so.

Figure 3.4. Define JDBC DataSource

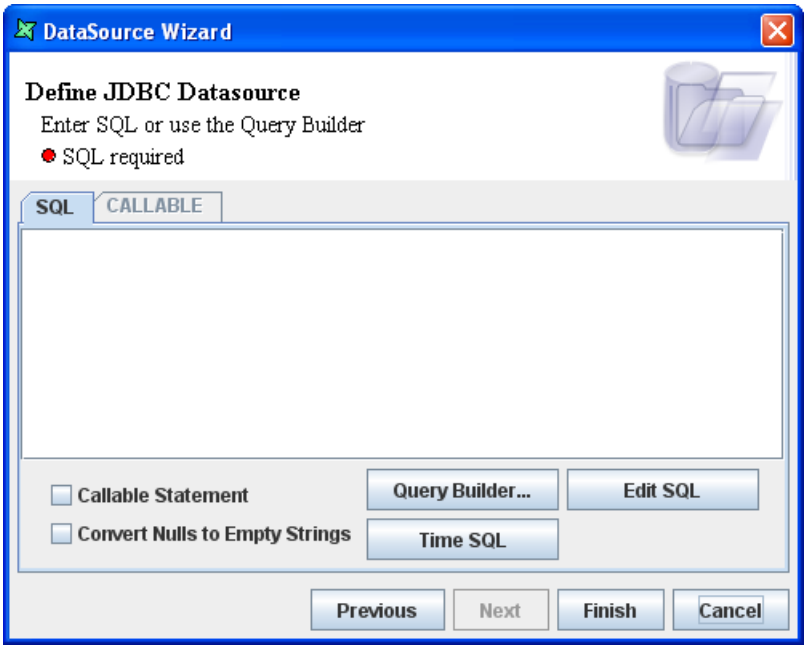
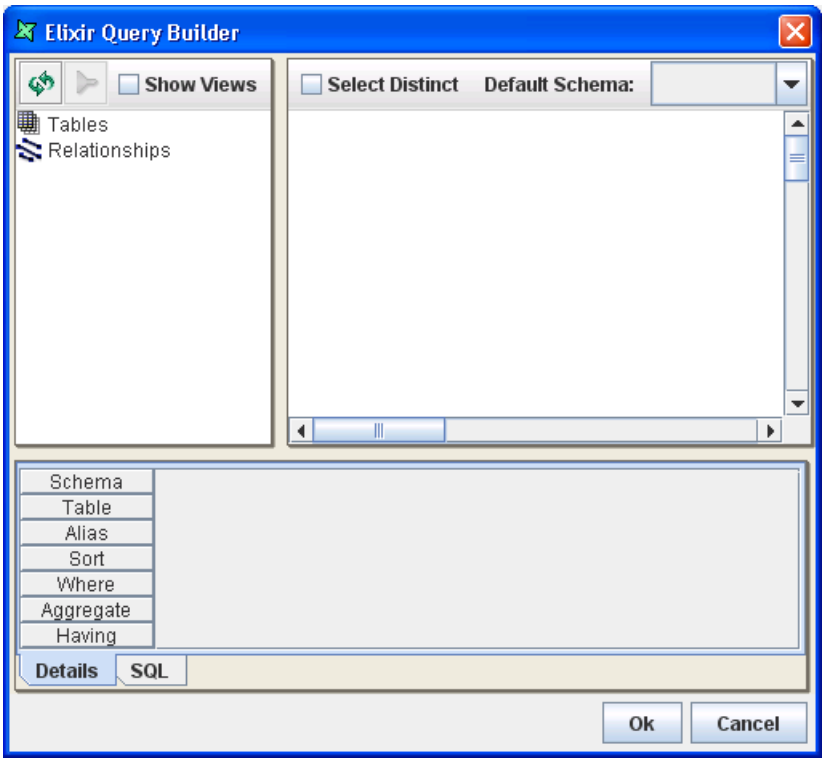


Figure 3.5. Elixir Query Builder

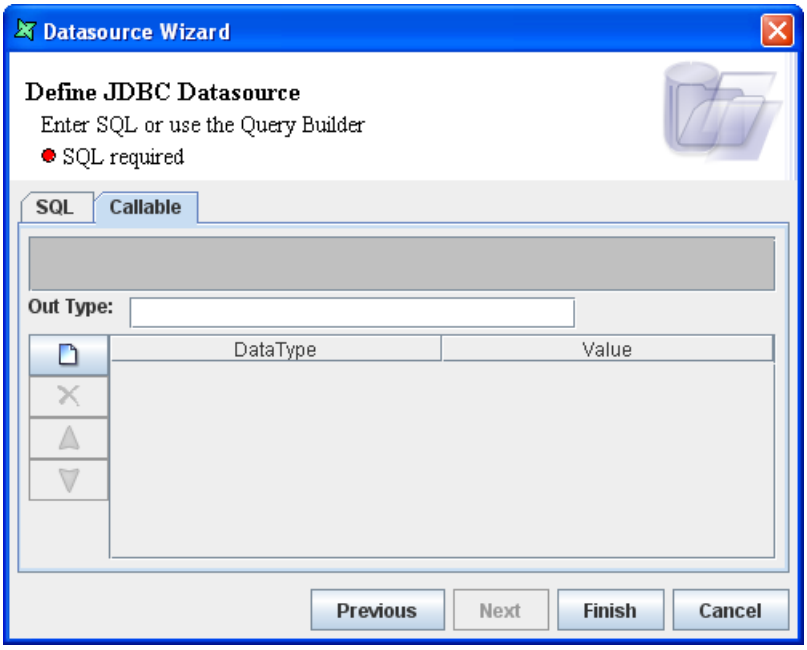


- Clicking the Query Builder button in the SQL window, opens the dialog box as shown in Figure 3.5, “Elixir Query Builder”.

- There are three panels in this dialog window. The panel on the left side lists the tables and relationships present in the database. There is a Show Views check box present on top of the left panel. When this check box is selected the views present in the database are listed. Tables and Views can be selected to form the basis of the query.

- In the right panel, fields of the selected tables and views are displayed. The fields to be included in the query can be selected. When the fields are selected the properties of the selected fields are displayed in the Details tab of the lower panel. Similarly, the field names are included in the SQL statement that is displayed in the SQL tab of the lower panel.
- Show button is enabled if there are multiple schemas in the database. From there, the user will be able to select specific schemas to be listed under the Query Builder dialog.
- If the "Select Distinct" check box on the right panel is checked then only distinct records of the table are selected - no duplicate records will be retrieved.
- Instead of using the Query Builder to create a SQL statement the SQL statement can be entered directly in the SQL window.
- Elixir Data Designer allows stored procedures to be invoked, including passing parameters to the database server (subject to database support). If you need to use a stored procedure, the callable statement syntax has to be entered in the SQL Window. Then the " Callable Statement" checkbox should be selected. On selecting the check box, the Callable tab is activated. In the Callable tab as shown in Figure 3.6, “Callable Tab”, the type of the output parameter must be specified.

Figure 3.6. Callable Tab



Note

If you are using the Remote version of Elixir Repertoire, remember that all data queries and operations are performed by the Elixir Repertoire Server. If you use "localhost" in your connection URL, localhost will refer to the server, not the client. Because all data operations are done on the server, you don't need a client-side version of the JDBC driver.

To add the JDBC driver to your system class path, find the location where Java is installed. This may be either the development kit (jdk), or runtime (jre). For the jdk there is directory jdk/jre/lib/ext, whereas for the jre it is just jre/lib/ext. Put the JDBC driver in this ext directory before launching the Remote tool. This step is only required if you want to use the Query Builder from the Remote tool. The JDBC-ODBC bridge provided by Sun is already included in the Java distribution and doesn't require any additional configuration.

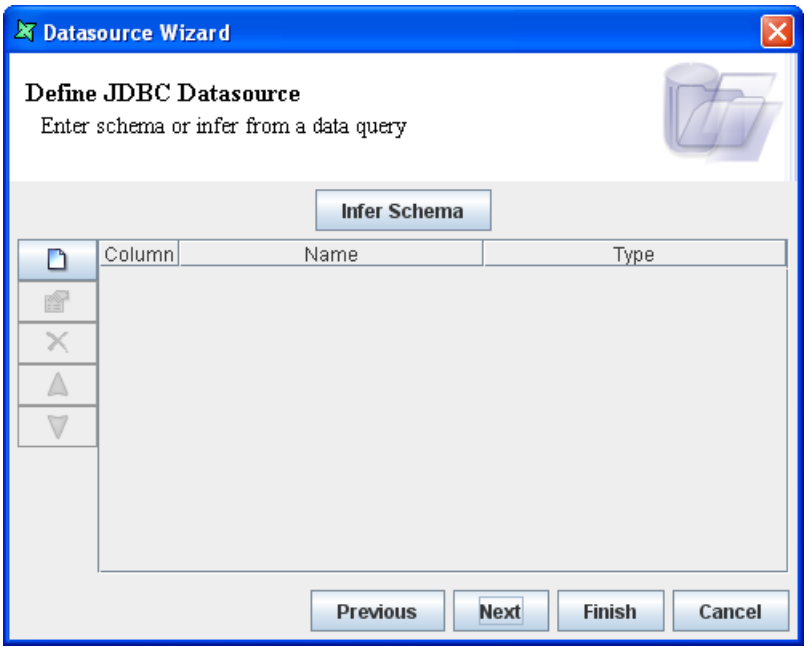
For JDBC datasource created with Repertoire 7.7 onwards, the Query Builder model will be retained after exiting from the wizard, unlike in older versions whereby user will need to create the model again when opening the wizard after exiting an exit. For JDBC datasource(s) created with Repertoire 7.6 and below, rebuild the model in Repertoire 7.7 and save it. The model will then be retained for future use.

Infer Schema

After entering the SQL or callable statement, click the Next button. The page appears as shown in Figure 3.7, “Define DataSource Schema”.

In this screen the schema can be inferred from the data query. Click the Infer Schema button. If a connection to the database can be made, the inferred fields and their corresponding data types will be listed.

Figure 3.7. Define DataSource Schema

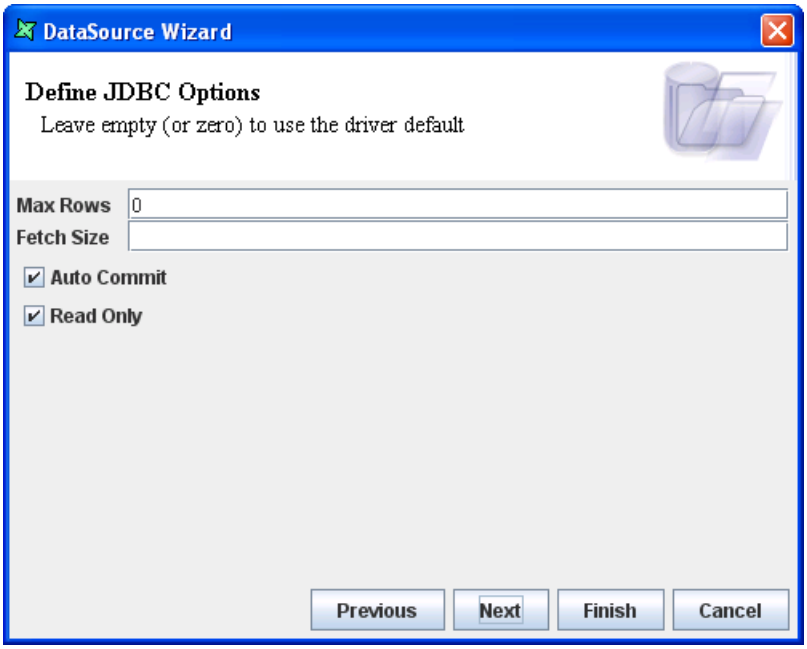


JDBC Options

Auto Commit: When checked, new record that is added will be updated accordingly.

Read Only: When checked, the datasource will only read from the database once. From then, the datasource will load data based on the data that was first read. Data will be retrieved at a faster speed as the driver assumes that no changes is made to the database.

Figure 3.8. Define DataSource Options

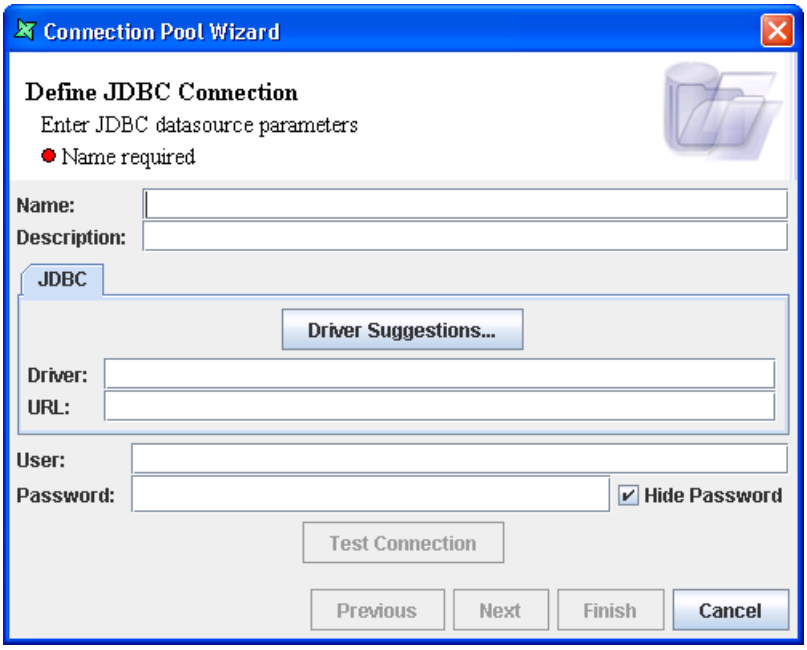


These 2 features are only available from Repertoire 7.3 onwards.

Connection Pools

Connecting to a relational database can be time-consuming, especially across a network. The job of a connection pool is to maintain a set of connections ready for use. You can create a connection pool in the Repository by choosing Add->Connection Pool... from a folder or filesystem popup menu. A wizard will appear, as shown in Figure 3.9, “Connection Pool Wizard”. The `Test Connection` button will only be enabled when the driver is being specified and will allow the user to test whether the connection to the database is available. A window will pop-up to indicate if the connection has succeeded or failed.

Figure 3.9. Connection Pool Wizard



The information required on the first page is exactly the same as that on the JDBC wizard described above. If you are creating a pool for use on a server, the warning "Driver class not found" means the class isn't found on the local classpath - the JDBC driver may already exist on the server. You can still proceed despite the warning and you will need to verify the server classpath (eg. the /ext directory of Elixir Repertoire Server) contains the necessary driver for your chosen database.

Note

Dynamic substitutions, such as ${password}, are not available in the Connection Pool because it may be shared by all users of the software and there is no common context from which the dynamic parameters could be obtained.

The next page of the Connection Pool Wizard allows the characteristics of the pool to be controlled. The Connection Pool is built upon c3p0 [http://www.mchange.com/projects/c3p0/index.html]

Max. Idle Time (s)	The time in seconds that a Connection can remain pooled but unused before being discarded. Zero means idle connections never expire.
Min. Pool Size	The minimum number of Connections the pool will maintain at any given time.
Max. Pool Size	The maximum number of Connections the pool will maintain at any given time.
Max. Statements	The size of the pool's global PreparedStatement cache. If Max Statements is zero, statement caching will be enabled, but no global limit will be enforced.
Max. Connection Age (s)	The time to live, in seconds. A Connection older than Max Connection Age will be destroyed and purged from the pool. This differs from Max Idle Time in that it refers to absolute age. Even a Connection which has not been much idle will be purged from the pool if it exceeds Max Connection Age. Zero means no maximum absolute age is enforced.
Acquire Increment	Determines how many connections at a time the pool will try to acquire when the existing connections in the pool are all in use.
Acquire Retry Attempt	Defines how many times the pool will try to acquire a new Connection from the database before giving up. If this value is less than or equal to zero, the pool will keep trying to fetch a Connection indefinitely.
Acquire Retry Delay (ms)	The time in milliseconds that the pool will wait between acquire attempts.

The final page of the Connection Pool Wizard allows the pool to be secured, by encrypting the file to provide some protection for the JDBC password.

Right-clicking on the .pool file will list a few options, like in Figure 3.10, “Right-click on Connection Pool”. One of them will be to create a JDBC datasource. Only the first screen of the wizard will be different as it will be using the Connection Pool to connect, similar to Figure 3.11, “Datasource Wizard”. The rest of the pages of the wizard will be the same as creating a JDBC Datasource using the `JDBC` tab.

Figure 3.10. Right-click on Connection Pool

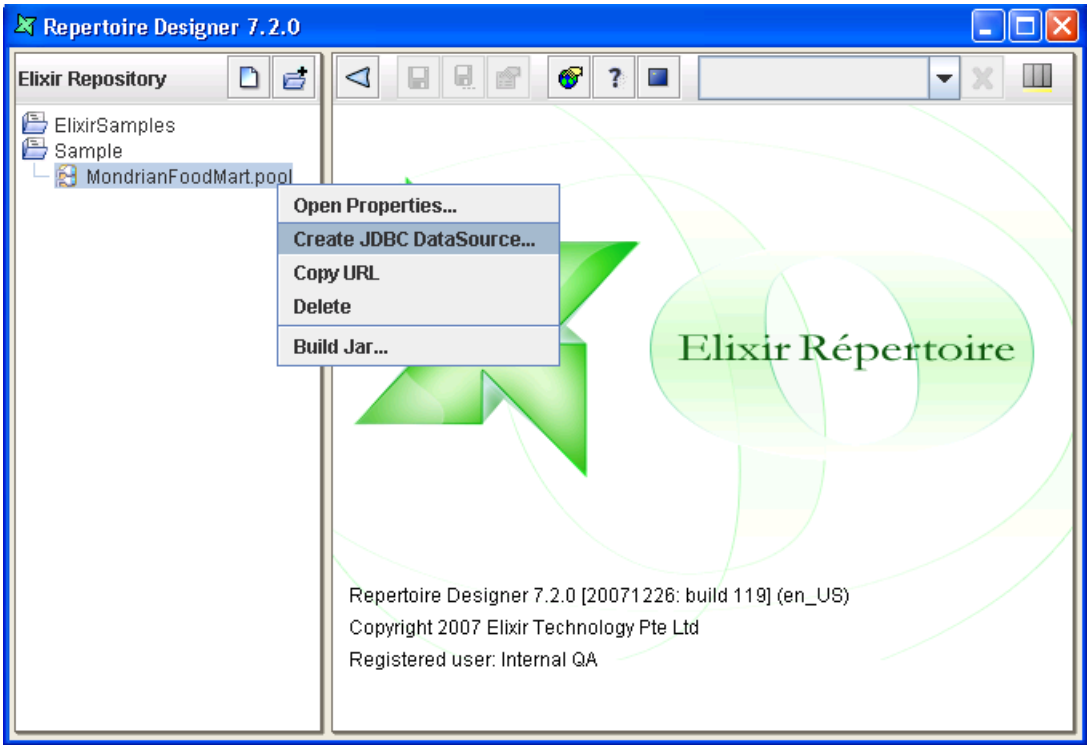
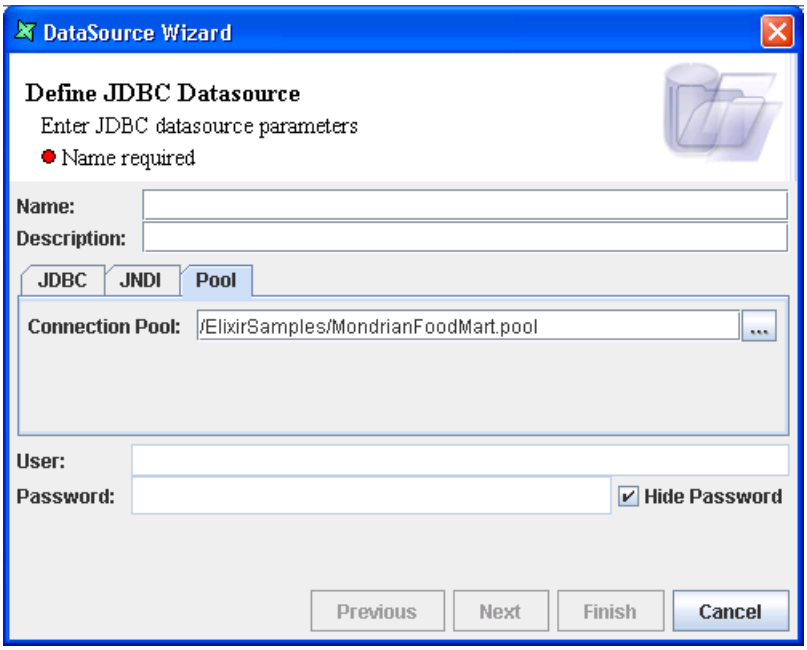


Figure 3.11. Datasource Wizard



Working with a JDBC DataSource

In this section we will review the different ways of adding a JDBC DataSource.

Using the JDBC/ODBC bridge driver

The JDBC/ODBC driver is only available on Microsoft Windows platforms and allows you to connect to Microsoft ODBC interfaces, such as Microsoft Access.

1. To add a Microsoft Access DataSource to the Elixir Repository we need to first register the ODBC source as described in the previous section. Select the Microsoft Access Driver(*.mdb) from the list of drivers. Enter the name `MondrianFoodMart` in the Data Source Name text box. Locate your copy of MondrianFoodMart.mdb and set the path accordingly.

2. Launch the Elixir Repertoire software.

3. Choose a file system or folder and Add a DataSource, choose JDBC and click the Next button.

4. Enter name `Sales` in the text box. Choose DataSource type JDBC.

5. By default the JDBC/ODBC_Bridge(Sun JVM) is selected as the Driver Suggestion. Enter the URL `jdbc:odbc:MondrianFoodMart` and the DataSource name as `Sales`. Click the Next button.

6. Click the Query Builder button. Select the Sales table from the list of tables and double click on it.

7. Select the Customer_id, Store_id and Store_sales fields from the table and click the OK button.

8. The query, including the selected columns, is displayed in the SQL window. Click the Next button.

9. Instead of using the Query Builder to build the SQL query, the query

   ```
   Select Customer_id, Store_id and Store_sales from sales
   ```

   can be entered directly in the SQL tab window. Click the Next button.

10. In the Define DataSource schema screen, click the Infer Schema button. The schema is inferred from the data query. Click the Finish button. The Sales.ds data source is added to the repository. The records in the data source can be viewed by clicking on the Load Data menu in the Data Window.

Note

Please refer to Appendix B, *Samples* for the sample files used in this chapter.

Using a Callable Statement

To make use of Stored procedure in the JDBC data source, a stored procedure must first be created in the database.

Here's how to create a stored procedure using Oracle:

1. Create a new table `emp` with columns Eno number, Dno number, Dname varchar2(12), and Esal number.

2. We want to create a stored procedure on the table to fetch records with specific value of Eno or Dno or Esal.

3. A package is created using the code below:

   ```
   CREATE OR REPLACE PACKAGE pack AS TYPE
   empRowType IS REF CURSOR return emp%rowtype; FUNCTION
   ```

```
selemp(enumber in number, dnumber in number, esalary in
number) RETURN empRowType; End pack;
```

Compile the package.

4. The package body is created using the following code:

```
CREATE OR REPLACE PACKAGE BODY pack AS
FUNCTION selemp(enumber in number, dnumber in number,
esalary in number) RETURN empRowType IS myemp
empRowType; BEGIN OPEN myemp for select * from emp
where (Eno=enumber) or (Dno=dnumber) or (Esal=esalary);
return myemp; End; End pack;
```

Compile the package body.

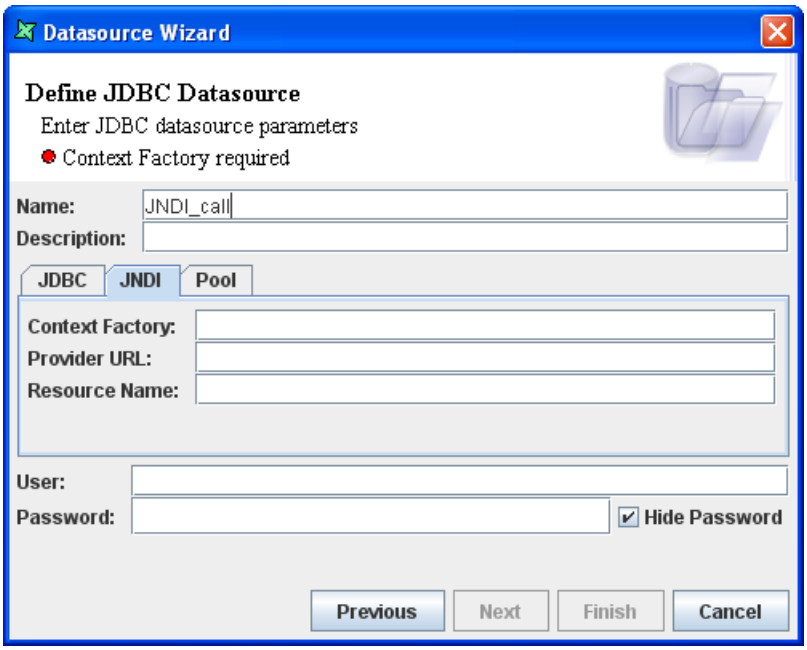
5. Make sure the Oracle driver classes12.jar is in the Elixir Repertoire ext folder. Launch Elixir Repertoire so that the driver is loaded. (Note if using the Remote software, the ext folder is on the server.)

6. Add a FileSystem `JDBC Stored1` using the procedure given in the previous chapter. Alternatively, the data source can be added to an existing FileSystem. Now select the folder, and choose Add -> DataSource from the popup menu. Select the JDBC DataSource type and click the Next button.

7. Enter the data source name `JDBC_Call`. Select Oracle(Thin_driver) as the driver suggestion. The driver and URL are assigned automatically. The URL can be altered according to the requirements. To make use of the stored procedure created in the Oracle server from the client system the IP address of the system has to be specified instead of localhost.

8. The user name and password are entered. After entering all the details, the JDBC data source screen of the DataSource wizard appears as shown in Figure 3.12, “JNDI Values”.

Figure 3.12. JNDI Values



9. Click the Next button. In the SQL window enter the syntax given below and select the IsCallable check box. The Callable procedure tab becomes active.

```
{?=call pack.selemp(?,?,?)}
```

Where pack is the package name and selemp is the name of the function.

10. Select the Callable tab.

    Specify the out type in the text box as given below

    ```
    oracle.jdbc.driver.OracleTypes.CURSOR
    ```

    The out type is database dependent. In this case a return cursor is specified as given above. Some databases do not need a return cursor, so the out type need not be specified for them. For more details on stored procedures with JDBC, refer to this article in JavaWorld. The link is

    http://www.javaworld.com/javaworld/jw-01-2000/jw-01-ssj-jdbc-p2.html

11. Click the Add button in the Callable tab and enter the Data type and values of the corresponding parameters. In this case three input parameters of integer Data type are needed.

12. Click the Next button and click Infer Schema to view the schema for the data source. Click the Finish button. The data source is added to the repository.

13. Open the JDBC_Call.ds data source. Click on the Load Data menu and verify that only the records with columns with the specified input values are fetched.

Using JNDI Connectivity

JNDI, the Java Naming and Directory Interface, allows applications to access various naming and directory services via a common interface. Like JDBC (Java Database Connectivity), JNDI is not a service, but a set of interfaces; it allows applications to access many different directory service providers using a standardized API.

JNDI uses the connection pooling technique to connect to JDBC datasources. Connection pooling is a technique that can be used to share database connections among requesting clients. When a connection has been created and is placed in a runtime object pool, an application can use that connection again. Each application does not have to perform the complete connection process every time it uses a connection.

When an application closes a connection, the connection is cached in the runtime object pool again. Connection pooling permits an application to use a connection from a pool of connections that do not have to be reestablished for each use. By using pooled connections, applications can realize significant performance gains because they don't have to perform all of the tasks that are involved in establishing a connection. This can be particularly significant for middle-tier applications that connect over a network, or for applications that repeatedly connect and disconnect..

The nodes in a JNDI namespace are known as contexts. The root node is known as the initial context. Initial contexts are created by initial context factories.

Elixir Data Designer provides enhanced support for JDBC DataSources deployed with JNDI.

The basic system requirements for using JNDI connections are:

- The vendor package providing connectivity for JNDI (Factory context) and the supporting classes.

Here's what to do:

1. Use the filesystem or folder popup menu and select Add -> DataSource.

2. Choose the JDBC datasource and click Next.

3. Select the JNDI tab.
4. Enter the context factory.
5. Enter the provider URL.
6. Enter the resource name. The resource points to the database.
7. Enter the user name and password if required.
8. The rest of the steps involved in connecting to the JDBC data source via JNDI using the Data Source wizard are similar to those followed in the above procedure using JDBC drivers.

Chapter 4
Composite DataSource

Overview

A Composite DataSource allows you to design a data flow using various data manipulation components in a network to get a personalized view of your data. Snapshots of the data can then be stored in various formats for further analysis.

Elixir Data Designer provides a visual interface that allows the interactive creation and manipulation of data. By using the various processing tools present in the data designer, aggregation and transformation of data can be performed easily. The output can be inspected at the end of every operation applied.

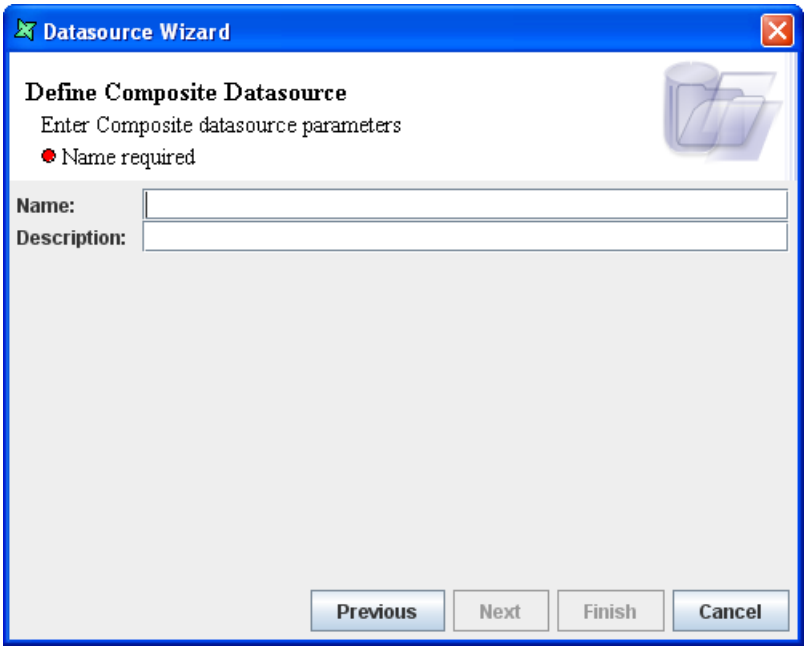
Adding a Composite DataSource

The Composite DataSource can be used to combine and control other DataSources. To add a Composite DataSource to a repository filesystem, you need to:

1. Select the FileSystem, choose Add-> DataSource from the popup menu and choose the Composite DataSource type then click the Next Button.

2. The "Define Composite DataSource" screen appears as shown in Figure 4.1, “Composite DataSource”. In this screen, enter the name of the Composite DataSource and description if any. Click the Finish button.

Figure 4.1. Composite DataSource

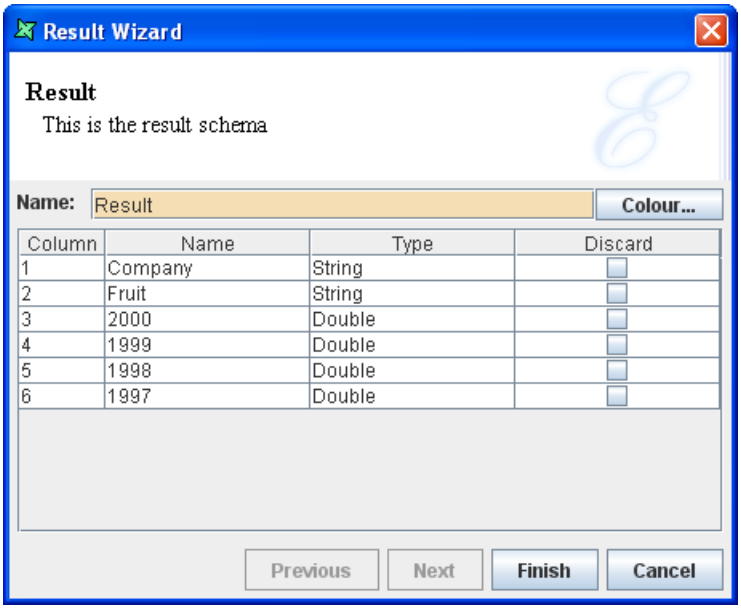


3. The Composite DataSource is added to the repository and opened. The Composite workspace will appear with three tabs: Designer, Script and Data.

In the Designer tab the Result box appears by default. The result of an operation performed in the data designer can be viewed using this graphic. This can be done by connecting the data output of any processor to the input of the Result box. When other tools, such as Elixir Report Designer connect to the data designer, they will use the data provided by Result.

All graphical shapes on the Composite diagram have a Popup menu. When the Properties menu item is selected from this menu, a Wizard will appear to allow editing of the item. The Result Wizard appears as shown in Figure 4.2, “Result Wizard”. You can also double-click on a shape to open the corresponding wizard.

Figure 4.2. Result Wizard



The name of the Result can be entered in the text box. On clicking the colour button the choose colour dialog box is invoked, from which the colour of the Result component can be selected. All of the graphical shapes allow editing of their names and colours in this way.

Standard Diagram Operations

Moving the processors

Shapes on the diagram can be moved by dragging them with the mouse. Alternatively, you can use the cursor keys to move the selected elements. Using a combination of Alt+cursor keys gives smaller increments.

Flow Connection Nodes

New nodes can be added to the flow by selecting Add Node from the flow's popup menu. You can add a number of nodes to your flows to neaten the appearance of your diagram.

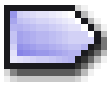
Data Processors

Data aggregation and transformation is performed by making use of the Data Operation processors. The Data Operation processors that are available in Composite designer environment are:

DataSource	Flow	Join	Sort	Derivative	Filter	Concat	Parameter
SubFlow	Note	Processor	DataDrop	Cube	DataStore		

Each processor and connector provides a popup menu and editable properties which can be accessed either through the popup menu, or by double-clicking the graphic on the diagram.

DataSource Processor



The DataSource processor forms the input to the designer environment. This processor, provides a diagrammatic representation of the different types of data sources available in the data designer.

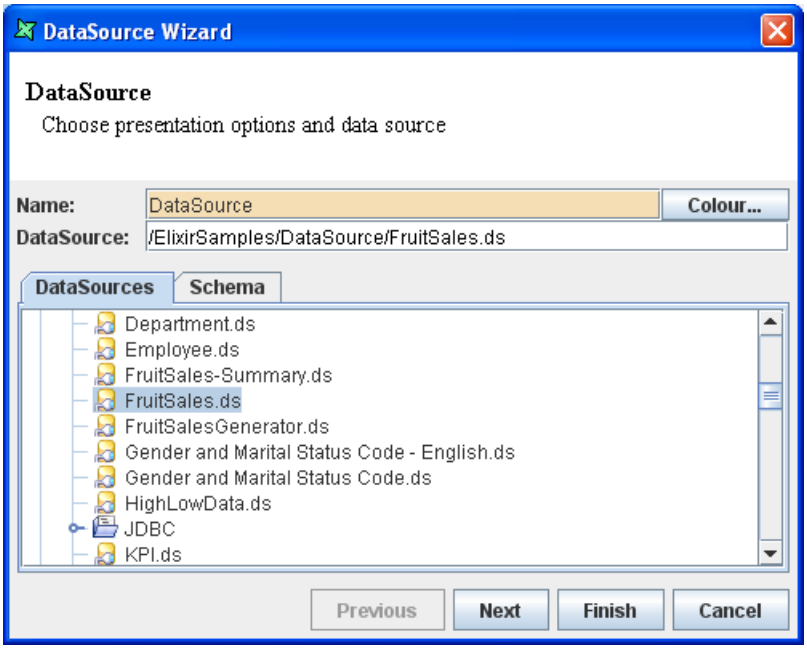
The data from another DataSource (it could be a primitive data source like JDBC or XML, or another Composite) can be extracted using this processor. It can be created in one of two ways:

- A DataSource in the repository can be selected, dragged and dropped on the designer diagram.
- Selecting the DataSource processor from the menu bar of the designer and clicking on the diagram creates a DataSource graphic which can then be connected to any repository datasource.

Properties

The editable properties are shown in Figure 4.3, “DataSource Wizard”.

Figure 4.3. DataSource Wizard



The data sources in the repository are listed in the DataSource window from which a data source can be selected.

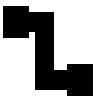
The name of the fields and the corresponding data type of the selected data source are displayed in the schema tab. The unwanted fields can be discarded by selecting the appropriate check box present in the Discard column. Discarding fields reduces the amount of memory needed to process the data.

Click the Next button to see any editable property values. Property values can be assigned here for any dynamic parameters in the selected data source. Refer to Appendix A, *Dynamic Parameters* for a detailed explanation on passing parameter values.

Note

Right-clicking on the Datasource processor shows the options available, one of which is to open the datasource. User will then be able to open the datasource directly instead of searching for it in the repository.

Flow



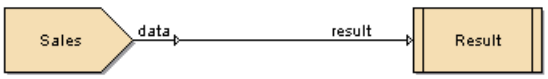
The Flow connector is used for linking the various processors to ensure a sequential flow of data.

If there is no valid flow of data, the flow connector appears with dashed lines. This happens when the flow is unable to read schema information from its input.

Here's how to use a flow processor to connect the Sales DataSource directly to the Result.

1. Add a JDBC DataSource `Sales` using the procedure given in the previous chapter.
2. Add a Composite DataSource named `Flow`.
3. Drag and drop the `Sales` DataSource in the Designer window.
4. Select the Flow control and connect the output of the `Sales` DataSource processor to the input of the Result. The data flow diagram appears as shown in Figure 4.4, “Sample Flow”.

Figure 4.4. Sample Flow



5. Use the View option on the Result popup menu to display the records in the Data tab.

User can inspect the values of respective fields by double-clicking on the flow and double-clicking on the desired field. Information like the total number of records, maximum value, minimum value and the average value of the data field are listed.

Join Processor

The Join processor is used to aggregate two data sources using one of several join types (e.g. Inner, outer, etc). The join operator links the records from two data sources by matching values in specified columns.

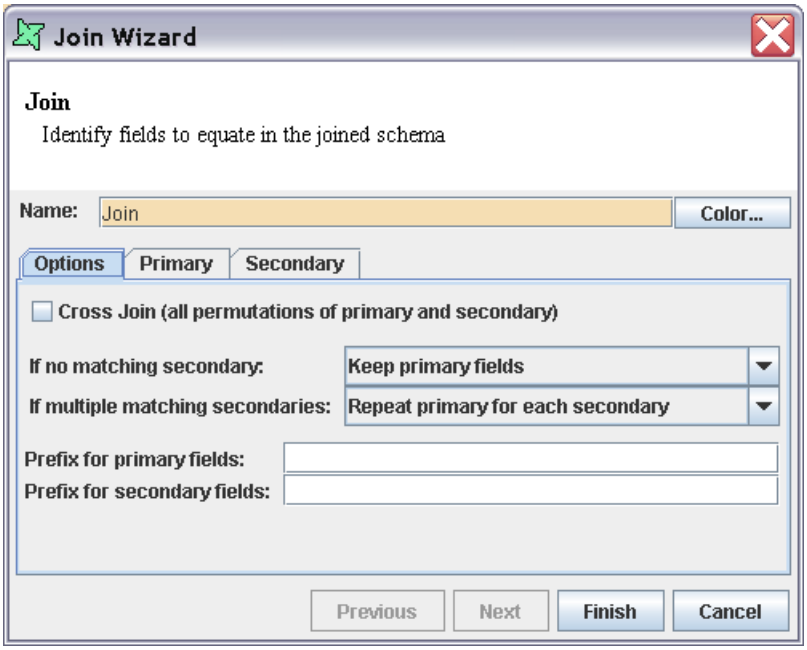
Elixir Data Designer supports all the three types of Joins: inner join, outer join and cross join.

The Join processor can be selected from the Designer menu bar and then placed on the diagram workspace. The Join processor must get input data from two DataSources.

Properties

The Join Processor properties are shown in Figure 4.5, “Join Wizard”.

Figure 4.5. Join Wizard



The name of the Join can be entered in the text box provided. The Choose Colour dialog box is invoked on clicking the Colour button from which the colour of the Join processor can be selected.

The dialog consists of three tabs: Options, Primary and Secondary.

- In the Options tab, a check box is provided for the Cross join. When this check box is selected the Cross Join is performed on the two data sources. The cross join produces every combination of input records. For example, if the primary datasource provides ten records and the secondary datasource provides five records, then fifty records (10x5) will be available as the output. For large datasets a huge number of records could be generated.

- There are two options Keep and Discard provided in the Combo box for "If no matching secondary". If the keep option is selected, then an Outer join can be performed on the data sources. If the Discard option is selected, then an Inner join can be performed on the data sources.

- Similarly, three options Repeat, Keep and Discard are provided for "If multiple matching secondaries". These control how multiple matches are handled. If the primary record matches three secondary records, the system can a) pass through three records, matching the single primary with each of the secondaries in turn (Repeat). b) just pass through the primary with just the first secondary (Keep), c) not pass through the record at all (Discard).

- The Primary prefix and the Secondary prefix can be entered in the text boxes if required. The Primary prefix and Secondary prefix are used to modify the field names in the merged output record based on the primary and secondary names in case they conflict.

  For example, an Employee DataSource with an Id and DepartmentId would conflict with a Department DataSource with its own Id. If both Ids need to be passed through, then one or both of them should be given a prefix to prevent duplicate names.

- The fields present in the primary data source are displayed in the Primary tab when the data source is connected with the Join processor . The fields of the primary data source that are not required can be discarded to reduce memory use.

- The fields present in the secondary data source are displayed in the Secondary tab when the data source is connected with the Join processor. The fields of the secondary data source that are not required can be discarded. Choose the appropriate key or keys in the secondary tab and associate them with the corresponding primary keys. Records will be selected from the secondary data source where the identified keys match those of the primary.

Working with Joins

Here's how to join the Sales and Customer data sources:

1. Add the JDBC data sources, Sales and Customer using the procedure given in the previous chapter.

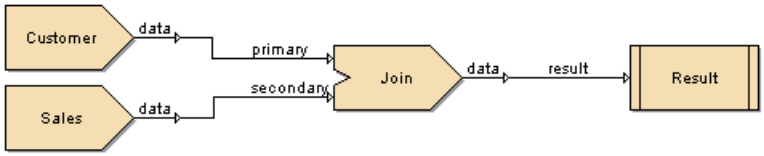
2. Add a Composite DataSource with the name `Join`.

3. Select the Sales data source, drag and drop it in the designer window. Similarly drag and drop the Customer data source onto the designer.

4. Create a Join processor on the diagram.

5. Connect the Customer data source as primary and Sales data source as secondary to the Join processor and connect the output to the Result. The diagram flow is shown in Figure 4.6, “Sample Join Flow”.

Figure 4.6. Sample Join Flow



Inner Join

Inner join combines column values from one record of a data source with column values of a record from another (or the same) data source to form a single, merged record of data.

Using an Inner join on the Sales and Customer data sources you can explore the result of setting different options for the "If multiple matching secondaries".

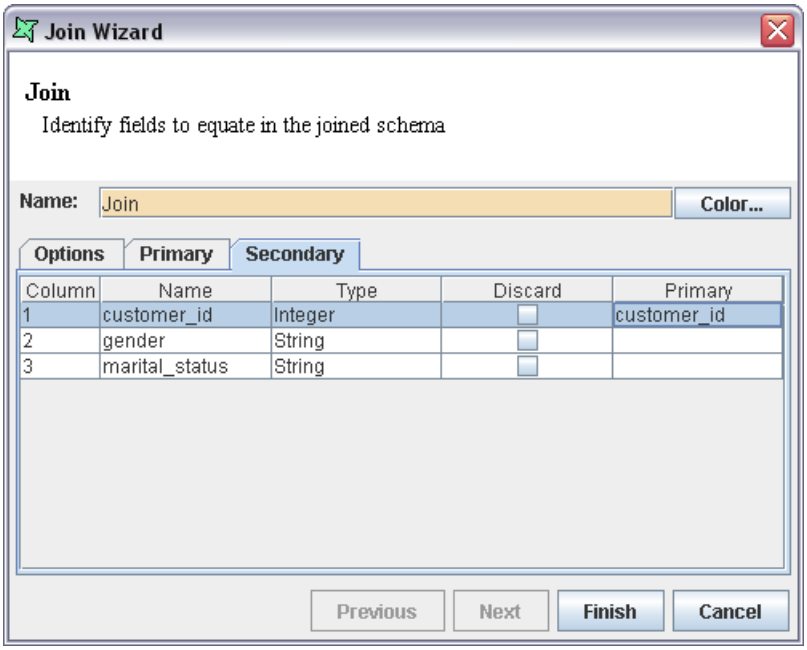
a) *If multiple matching secondaries - Repeat*

1. Using the flow described above, open the Join properties and select the `Discard` option from the "If no matching secondary" Combo box.

2. Select the `Repeat` option from the "If multiple matching secondaries" combo box.

3. Select the secondary tab. Select the customer_id field in the primary column against the customer_id field of the secondary data source as shown in Figure 4.7, “Join Wizard”.

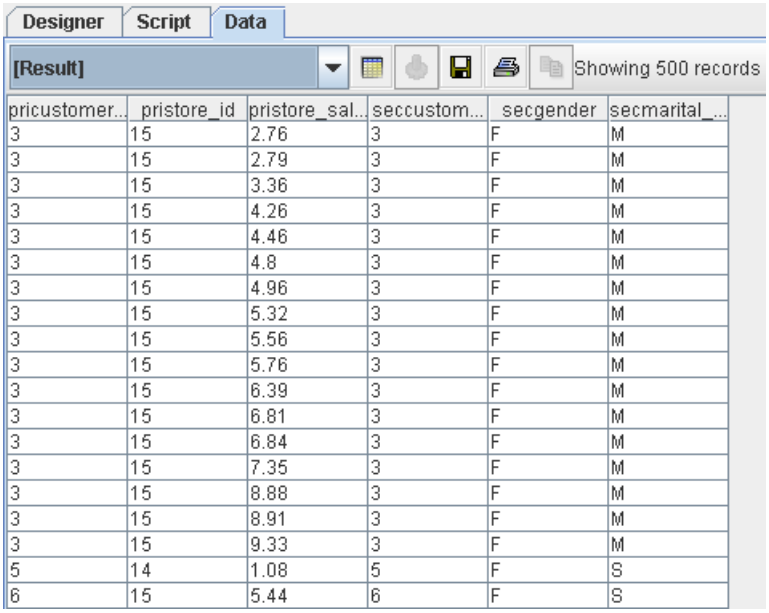
Figure 4.7. Join Wizard



4. Click the Finish button to close the Join Wizard and return back to the Designer window.

5. Select the Result, and choose View from the popup menu. The output is shown in the Figure 4.8, “Inner Join Result”.

Figure 4.8. Inner Join Result



pricustomer...	pristore_id	pristore_sal...	seccustom...	secgender	secmarital_...
3	15	2.76	3	F	M
3	15	2.79	3	F	M
3	15	3.36	3	F	M
3	15	4.26	3	F	M
3	15	4.46	3	F	M
3	15	4.8	3	F	M
3	15	4.96	3	F	M
3	15	5.32	3	F	M
3	15	5.56	3	F	M
3	15	5.76	3	F	M
3	15	6.39	3	F	M
3	15	6.81	3	F	M
3	15	6.84	3	F	M
3	15	7.35	3	F	M
3	15	8.88	3	F	M
3	15	8.91	3	F	M
3	15	9.33	3	F	M
5	14	1.08	5	F	S
6	15	5.44	6	F	S

This query fetches all primary records which have one or more matching secondary records. The primary record is repeated once for each secondary match. If there is no secondary match, the primary record is discarded.

b) *If multiple matching secondaries - Keep*

This query fetches all primary records which have one or more matching secondary records. The primary record is only output once, merged with the first secondary match. Subsequent secondary matches are ignored. If there is no secondary match, the primary record is discarded.

c) *If multiple matching secondaries - Discard*

This query fetches all primary records which have exactly one matching secondary record. The primary record is output once merged with the secondary match. If there is no secondary match, or multiple secondary matches, the primary record is discarded.

Outer Join

Outer join is a type of join in which both matching and non matching rows are returned. The values of all columns from the unmatched table in non-matching rows are set to NULL.

Using an Outer join on the Sales and Customer data sources you can explore the result of setting different options for the "If multiple matching secondaries".

a) *If multiple matching secondaries - Repeat*

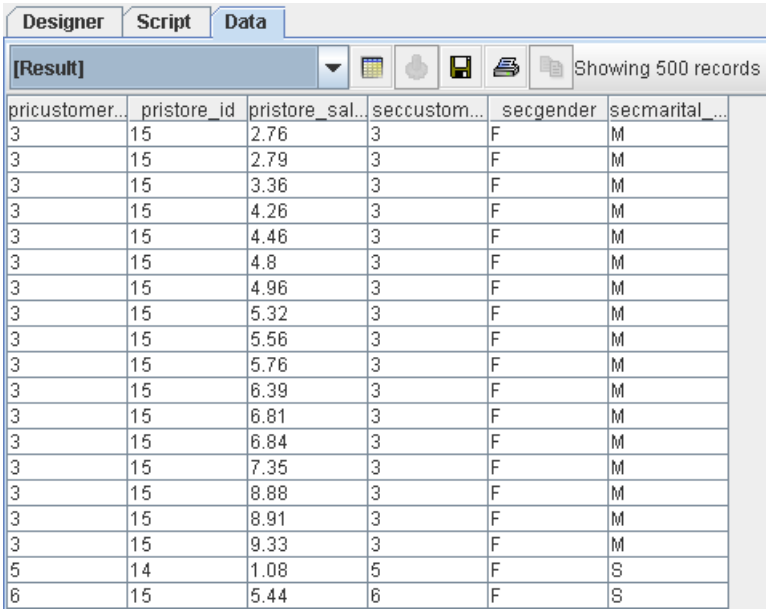
1. Return to the same `Join` diagram and edit the Join properties by selecting the option `Keep` from the "If no matching secondary" and `Repeat` from the "If multiple matching secondaries" Combo box.

2. Enter "pri" in the Primary Prefix text box and "sec" in the Secondary Prefix text box.

3. We will leave the secondary tab as before, linking the customer_id fields between primary and secondary.

4. Click the Finish button and view the Result output. The output is shown in the Figure 4.9, “Outer Join Result”.

Figure 4.9. Outer Join Result



pricustomer...	pristore_id	pristore_sal...	seccustom...	secgender	secmarital_...
3	15	2.76	3	F	M
3	15	2.79	3	F	M
3	15	3.36	3	F	M
3	15	4.26	3	F	M
3	15	4.46	3	F	M
3	15	4.8	3	F	M
3	15	4.96	3	F	M
3	15	5.32	3	F	M
3	15	5.56	3	F	M
3	15	5.76	3	F	M
3	15	6.39	3	F	M
3	15	6.81	3	F	M
3	15	6.84	3	F	M
3	15	7.35	3	F	M
3	15	8.88	3	F	M
3	15	8.91	3	F	M
3	15	9.33	3	F	M
5	14	1.08	5	F	S
6	15	5.44	6	F	S

This query fetches all primary records. Each primary will be output as many times as there are matching secondary records. However, if there is no matching secondary record, the primary record will still be output once with the secondary fields set to NULL.

b) *If multiple matching secondaries - Keep*

This query fetches all records from the primary datasource and connects them to the first matching record from the secondary datasource. If no secondary record matches, the output record contains NULL for those values.

c) *If multiple matching secondaries - Discard*

This query fetches all records from the primary datasource. If there is exactly one matching secondary, it is merged. If there is zero or more than one matching secondary then NULL is used for each secondary field in the output.

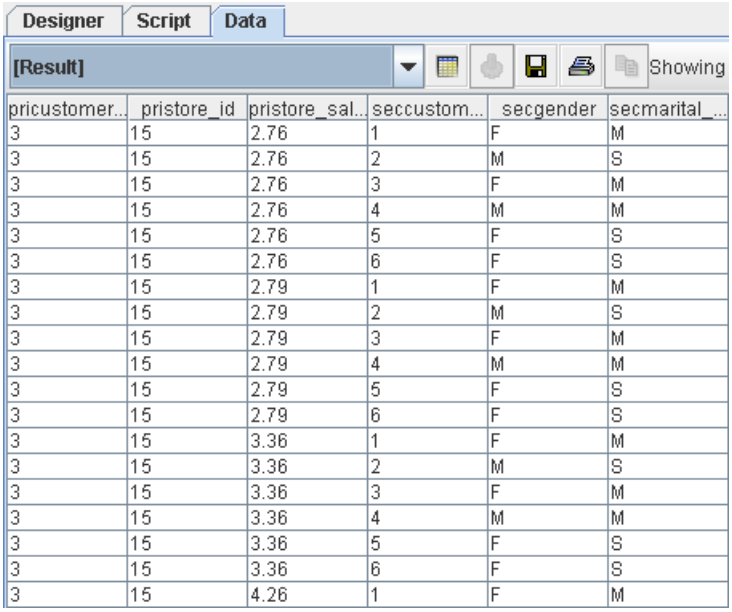
Cross Join

A Cross Join merges each possible combination of records from the primary and secondary datasources. The output from a cross join can be very large!

1. Reusing the same `Join` composite, open the Properties and select the Cross Join check box. You will notice the combo boxes are now disabled as all combinations are generated.

2. Click the Finish button and view the Result output. The output is shown in the Figure 4.10, “Cross Join Result”.

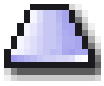
Figure 4.10. Cross Join Result



pricustomer...	pristore_id	pristore_sal...	seccustom...	secgender	secmarital_...
3	15	2.76	1	F	M
3	15	2.76	2	M	S
3	15	2.76	3	F	M
3	15	2.76	4	M	M
3	15	2.76	5	F	S
3	15	2.76	6	F	S
3	15	2.79	1	F	M
3	15	2.79	2	M	S
3	15	2.79	3	F	M
3	15	2.79	4	M	M
3	15	2.79	5	F	S
3	15	2.79	6	F	S
3	15	3.36	1	F	M
3	15	3.36	2	M	S
3	15	3.36	3	F	M
3	15	3.36	4	M	M
3	15	3.36	5	F	S
3	15	3.36	6	F	S
3	15	4.26	1	F	M

The combination of all the records from both data sources are displayed. For instance, if there are 25 records in the Customer table and 40 records in the Sales table a total of 1000 (25*40) records are fetched by using Cross Join.

Sort Processor



The Sort processor is used to sort data source records using specified criteria in the ascending or descending order. It includes a Group On option to group the records depending on their values or

specified ranges. Unlimited levels of sorting and grouping can be done on a data set. The Group Range used for delimiting groups can be specified for a field of type number, date or string.

Elixir Data Designer provides an Expression Builder to assist in the extraction of ranges of records. Using this, the top n or bottom m records present in the data source can be retained. These values need not be constants, they can be variables and also include percentages.

The Sort processor is selected from the menu bar of the Designer window and then placed in the designer window workspace.

Properties

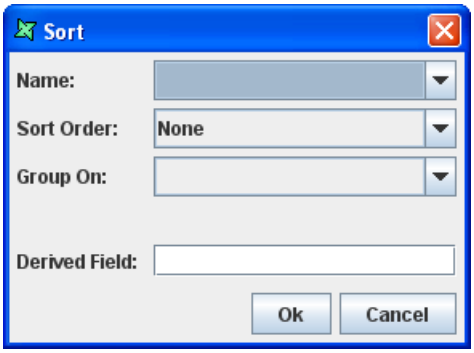
The editable properties are shown in Figure 4.11, “Sort Wizard”.

Figure 4.11. Sort Wizard

There are four columns in the sort table: Name, Sort Order, Group On and Group Data Field. The columns will be sorted based on the values assigned to these columns.

On clicking the Add button, a Sort dialog pops up as shown in Figure 4.12, “Add Sort Item”. The following options are available in the Sort dialog box:

Figure 4.12. Add Sort Item



- In the "Name" Combo box the field name of the data source based on which the sorting has to be done is selected.

- The Sorting order as none, ascending or descending has to selected from the "Sort Order" combo box. There are two new sort orders, simple ascending and descending sort. These uses a more simplistic algorithm is to compare unicode character values literally. The benefit of this is that it doesn't ignore spaces.
- The options available in the Group On combo box is based on the data type of the selected field. The Group On options are discussed below.
- Finally, on clicking the OK button the Sort options specified in the Sort Wizard are added in the corresponding columns.

On clicking the Next button, the "Extract screen appears as shown in Figure 4.13, “Extract Options”. The main part of this screen is the Expression Builder.

Figure 4.13. Extract Options

- The Top option is selected if the top n number of records has to be retained.
- The Bottom option is selected to retain the bottom n number of records.
- In the Amount field, the value of 'n' has to be specified. Where 'n' is the number of records to be sorted.
- If the Percent check box is selected then the Amount value represents a percentage of the total number of records flowing through.
- The level field indicates the level at which the extraction algorithm applies. A value of 0 indicates that it applies over all records. A value of 1 indicates the top n records of each group level 1 will be retained. The value can vary between 0 and the number of grouping levels identified on the previous page
- Having edited all the values, clicking the Set Value button will produce a command string in the Extract Text box. This string can be edited manually, for example to include dynamic parameters.

Group On Options

The Group On combo box lists the various options available for each field based on its data type.

Numerical Data

If a number type is selected for grouping then the options that are listed in the "Group On" combo box are Each Value, Range, Count and All.

String Data

If the String data type field is selected for grouping then the options that are listed in the "Group On" Combo box are Each Value, Substring, Range, Count and All.

Date Data

If the Date data type field is selected for grouping then the options that are listed in the "Group On" combo box are All, Count, Each Value, Year, Quarter, Month, Week of Year, Week of Month, Day of Year, Day of Month, Day of Week, Range.

- *Each Value* - When this option is selected then consecutive records (after sorting) with the same selected field value are grouped together.
- *All* - When this option is selected then all the records are grouped together irrespective of the selected field value
- *Count* - When this option is selected then the records are grouped with a fixed number of records in each group. For example, 20 records with Count=3 would yield 7 groups. The first six groups would contain three records each (18 in total) and the last group would only contain two.
- *Range* - When this option is selected the range text field appears. The syntax for specifying the range is [condition]:label, for example "<x:A". Where x is the threshold value based on which grouping is done and A is the label name that is assigned for the group. A number of ranges can be specified. For example:

  ```
  <5:A|<10:B|<15:C
  ```

  A, B and C are the label names assigned for those groups. Alternate conditions can be described by using operators, '>' and '='. Ranges are always tested left-to-right and the first one that matches is chosen. You can leave a condition off the last range expression to form the default value:

  ```
  =A:First|<E:Pass|:Fail
  ```

  This example will return 'First' for input 'A', 'Pass' for inputs like 'B','C','D' and 'Fail' for all other inputs.
- *Substring* - When this option is selected from the list then the [Start,End] text box appears. In this text box the starting and ending character of the string is specified. For instance, if the syntax specified in the text box is [0,3] then the grouping will be based on 3 characters starting from the first character of the string (indexes are zero-based).

  There is also an option to add a derived field that holds the substring values, so that subsequent processors can use them without having to derive them again. To use this feature, just enter a unique name for Derived Field. If you inspect the output schema of the Sort, you will see the additional field is listed.
- *Year* - When this option is selected and a value is entered in the Group Interval text box the records are grouped according to the Years of the specified field and the group interval. Suppose the years in the date field are starting from 2000-2020 and the "Year" option is selected from the Group On combo box and the value 2 is entered as group interval, then the records containing years 2000-2001, 2002-2003, 2004-2005, etc are grouped together.
- *Month* - When this option is selected and a value is specified in the Group Interval text box the records are grouped based on the Month of the specified field and the group interval. For instance,

if the Month is selected as the Group On option and the Group Interval is entered as 3 the months are sorted as January-March, April-June, etc irrespective of the years. If suppose, the Month grouping has to be done for a particular group interval of years, then the years must be grouped followed by the grouping of months.

- *Quarter* - When this option is selected and a value is entered in the Group Interval text box the records are grouped based on the quarter function of the specified field. For instance, if the Quarter is selected as the Group On option then the records are categorized into four quarters. The first quarter is represented as Q01: January-March. The second quarter is represented as Q11: April-June. The third quarter is represented as Q21:July-September and the fourth quarter is represented as Q31: October-December. If suppose, the quarter grouping has to be done for a particular interval of years then the years must be grouped followed by the quarter grouping.

- *Week of Year* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the number of weeks in a year and the group interval. Since, there are a total of about 52 weeks in a year the grouping is done accordingly. For instance, if the Week of Year is selected as the Group On option and the value of group interval is entered as 3 then the grouping output is 0-2, 3-5, etc. Where 0-2 corresponds to the first two weeks of the years as 0 corresponds to a null value and 3-5 includes the next three weeks of the year, so on. If suppose, the Week of the Year grouping has to be done for a particular interval of year then the years must be grouped first followed by the week of the year grouping. Sunday is always considered as the first day of a week while using Week of Month option.

- *Week of Month* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the Week of the month and the group interval. For instance, if the Week of Month is selected as the Group On option and the value of group interval is entered as 2 then the grouping output is 0-1, 2-3, 4-5, etc. Where 0-1 fetches only the first week of the month since 0 corresponds to a null value and 2-3 groups the second and third week of the month. If suppose, the Week of the month grouping has to be done for a particular interval of year and month then the years must be grouped first followed by months. Finally, the week of the month grouping is done. Sunday is always considered as the first day of the week while using Week of Month option.

- *Day of Year* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the record are grouped based on the days of the year and the group interval. Since there are 365 days in a year the grouping is done accordingly. For instance, if the Days of Year is option is selected and the group interval is specified as 4 then the grouping output is 0-3, 4-7, etc. Where 0-3 fetches the records corresponding to the first 3 days of the year as 0 corresponds to null value and 4-7 the fourth to seventh day of the year. If suppose the day of the year grouping has to be done for a particular interval of year then the years must be grouped first followed by the days of the year grouping.

- *Day of Month* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the days of the month and the group interval. For instance, if the Day of Month option is selected and the group interval is specified as 1 then the grouping output is 1,2...30 or 31 depending on the total number of days in the month. If suppose, day of the month grouping has to be done for a particular interval of the months of a year then the years are grouped first by the month grouping. Finally the Day of Month grouping is done.

- *Day of Week* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the days of the week and the group interval. For instance, if the Day of Week option is selected and the group interval is entered as 2 then grouping output is Monday-Tuesday, Wednesday-Thursday, Friday- Saturday and finally Sunday. That is the records corresponding to the fields having Monday and Tuesday as the days of the week are grouped together and so on. If suppose the day of the week grouping has to be done for a particular interval of year and months then the years are grouped first followed by the month grouping. Finally, the Day of Week grouping is done.

The Hour, Minute and Second options are listed in the combo box only if the date field selected for grouping contains time or if the time field is selected for grouping.

- *Hour* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the hours. For instance, if the Hour Group On option is selected and the group interval is specified as 3 then the output is 0-2, 3-5, etc. Where 0-2 returns the records corresponding to the first 3 hours of the day starting from the 12th hour and 3-5 groups the next three hours of the day.
- *Minute* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the minutes. As there are 60 minutes in an hour the grouping is done accordingly. For instance, if the Minute Group On option is selected and the group interval is specified as 4 then the output is 0-3, 4-7, etc. Where 0-3 returns the records corresponding to first 4 minutes of the hour starting from the 0th minute and 4-7 returns the records having fourth to seventh minute of the hour.
- *Second* - When this option is selected and a value is entered in the Group Interval text box for a particular field then the records are grouped based on the seconds. As there are 60 seconds in a minute the grouping is done accordingly. For instance, if the Second Group On option is selected and the group interval is specified as 0-3, 4-7, etc. Where 0-3 returns the records corresponding to the first 4 seconds of the minute starting from the 0th second and 4-7 returns the records having fourth to seventh second of the hour.

Working With Sort processors

1. Add the JDBC data source for Sales using the procedure given in the previous chapter.
2. Add a Composite DataSource named `Sort`.
3. Now select the Sales data source, drag and drop it on the Composite diagram.
4. Place a Sort processor on the diagram.
5. Connect the Sales data source to the Sort processor and the output of the Sort to the Result. The diagram should appear as shown in Figure 4.14, “Sample Sort Flow”.

Figure 4.14. Sample Sort Flow

Sorting records by applying Sort Order Options

Ascending

1. Open the Sort processor properties.
2. In the first screen of the Sort Wizard click the Add button. The Sort dialog box appears.
3. Select `store_sales` from the list of field names available in the Combo box.
4. Select the Sort Order as Ascending from the Combo box. Click the OK button. The sort column is added to the Sort Wizard.

5. Click Finish button and view the Result to see all the records are sorted in ascending order by store_sales.

Extracting records using Expression Builder

a) Top n records

1. Open the Sort properties and remove any previous sort options (from the previous examples).
2. In the first screen of the Sort Wizard click the Add button. The Sort dialog box appears.
3. Select the store_id from the list of field names available in the Combo box.
4. Select Descending Sort Order and click the OK button. The sort column is added to the Sort Wizard. On clicking the Next button, the extract screen appears.
5. In the extract screen, select the top option and enter 6 in the Amount field.
6. Click the Set Value button. The values entered in the fields of the expression builder will get displayed in the text box.
7. Click the Finish button and view the records produced - the records are sorted in descending order by store_id and the top six retained.

b) Percentage of bottom n records

1. Follow the same procedure as a) above, except in the extract screen, select the bottom option and enter 25 in the Amount field. Select the Percent check box.
2. Click the Set Value button. The values entered in the fields of the expression builder will get displayed in the text box.

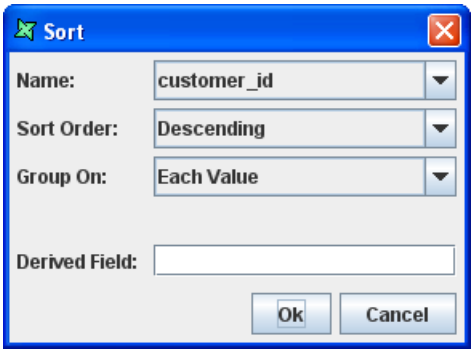
With this modification, the records are still sorted in descending order, but now the bottom 25% of the records are retained. That is out of 24 records only 6 records are displayed.

Sorting records by applying Group On Options

a) *Group On - Each Value*

1. Using the same sort diagram as the previous section, open the Sort properties and remove any previous sort and extract options.
2. In the first screen of the Sort Wizard click the Add button. The Sort dialog box appears.
3. Select the customer_id from the list of field names available in the Combo box.
4. Select Descending Sort Order.
5. Select the Each Value option from the Group On Combo box. The dialog box appears as shown in Figure 4.15, “Sort Dialog”. On clicking the Ok button, the sort column gets added to the Sort Wizard.

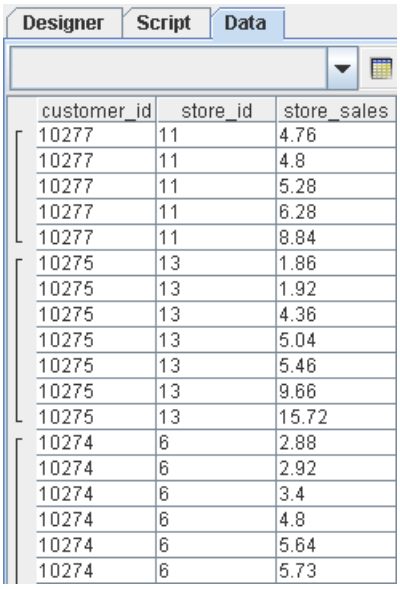
Figure 4.15. Sort Dialog



6. Click the Finish button in the Sort Wizard and view the Result.

The output is shown in Figure 4.16, “Group on - Each Result”. Records with the same consecutive values of customer_id are grouped together in descending order.

Figure 4.16. Group on - Each Result



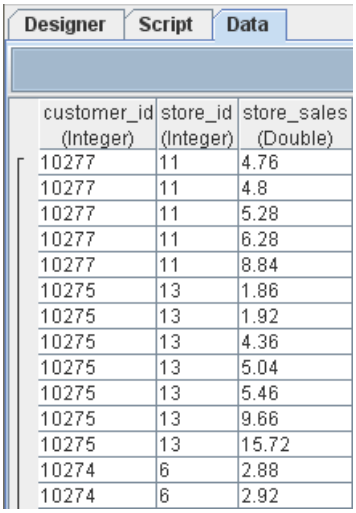
customer_id	store_id	store_sales
10277	11	4.76
10277	11	4.8
10277	11	5.28
10277	11	6.28
10277	11	8.84
10275	13	1.86
10275	13	1.92
10275	13	4.36
10275	13	5.04
10275	13	5.46
10275	13	9.66
10275	13	15.72
10274	6	2.88
10274	6	2.92
10274	6	3.4
10274	6	4.8
10274	6	5.64
10274	6	5.73

b) *Group On - All*

1. Modify the previous example by selecting the All option from the Group On Combo box.

The output is shown in Figure 4.17, “Group on - All Result”. It is seen that all the records irrespective of the value of customer_id are grouped together in descending order. This form of grouping is useful when you want to perform subsequent group operations using other tools in the Elixir toolchain - for example, a group-based chart can show results collected from all records.

Figure 4.17. Group on - All Result



customer_id (Integer)	store_id (Integer)	store_sales (Double)
10277	11	4.76
10277	11	4.8
10277	11	5.28
10277	11	6.28
10277	11	8.84
10275	13	1.86
10275	13	1.92
10275	13	4.36
10275	13	5.04
10275	13	5.46
10275	13	9.66
10275	13	15.72
10274	6	2.88
10274	6	2.92

c) *Group On - Range*

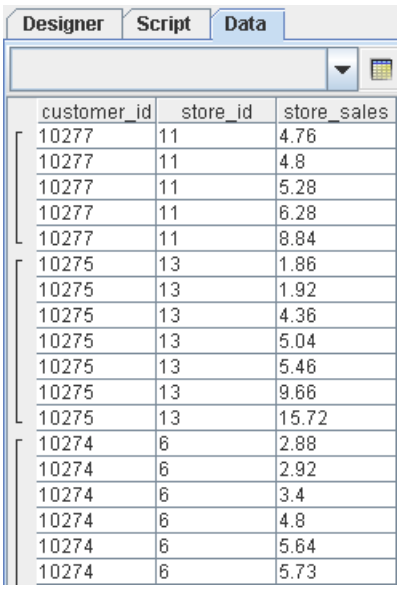
Numeric data

1. Modify the previous example by selecting the Range option from the Group On Combo box. Enter the value given below in the Range text box.

```
<=10274:a|<=10275:b|<=10277:c
```

The output is shown in Figure 4.18, “Group on - Range Result”. The records are grouped into a, b and c based on the values of the customer_id in ascending order. Where 'a' group includes all records less than or equal to 10274, 'b' group includes all the records less than or equal to 10275 and the 'c' group includes all the records less than or equal to 10277.

Figure 4.18. Group on - Range Result



customer_id	store_id	store_sales
10277	11	4.76
10277	11	4.8
10277	11	5.28
10277	11	6.28
10277	11	8.84
10275	13	1.86
10275	13	1.92
10275	13	4.36
10275	13	5.04
10275	13	5.46
10275	13	9.66
10275	13	15.72
10274	6	2.88
10274	6	2.92
10274	6	3.4
10274	6	4.8
10274	6	5.64
10274	6	5.73

String data

1. Instead of the Sales data source specified in the sort procedure given above, drag and drop the Stores data source and connect it to the Sort processor. As usual, the Sort processor output is connected to Result.

2. Open the Sort properties and in the first screen click the Add button. The Sort dialog box appears.

3. Select the Country from the list of field names available in the Combo box.

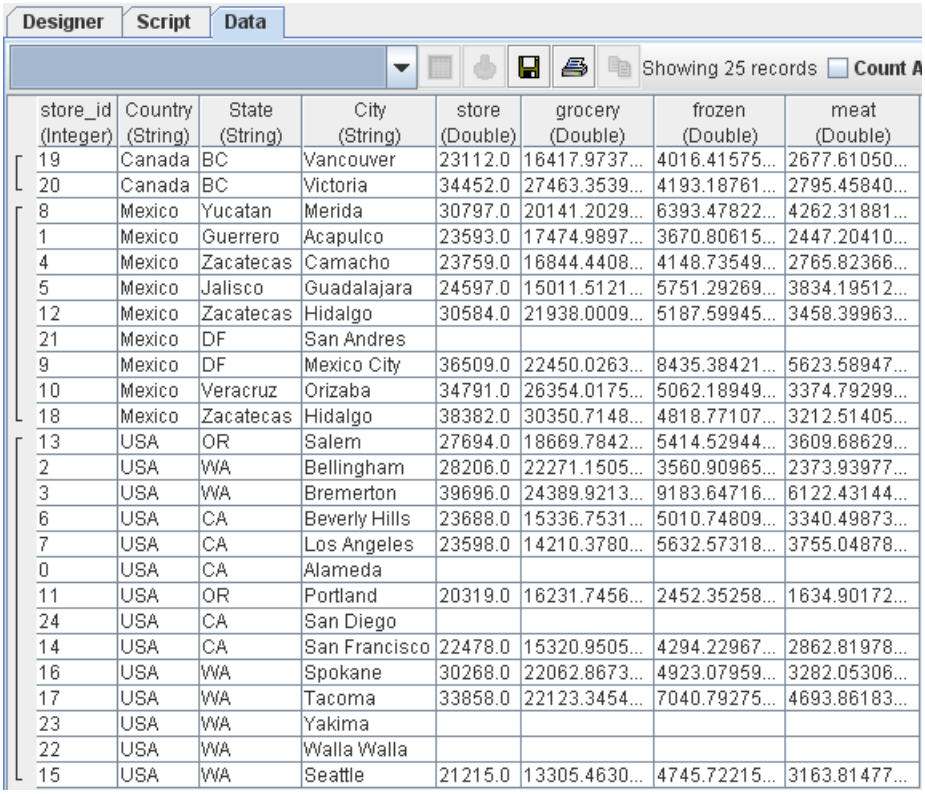
4. Select the Sort Order as "Ascending" from the Combo box.

5. Select the Range as the Group on option. Enter the value given below in the Range text box.

```
=USA:a|=Mexico:b|=Canada:c
```

The output is shown in Figure 4.19, “Group on - Range by Country Result”. It is seen that the records are grouped into a, b and c based on the values of the Country in ascending order. Where 'a' group includes all the records having Country name as USA, 'b' group includes records with Country name Mexico and 'c' group records with Country name Canada.

Figure 4.19. Group on - Range by Country Result



store_id (Integer)	Country (String)	State (String)	City (String)	store (Double)	grocery (Double)	frozen (Double)	meat (Double)
19	Canada	BC	Vancouver	23112.0	16417.9737...	4016.41575...	2677.61050...
20	Canada	BC	Victoria	34452.0	27463.3539...	4193.18761...	2795.45840...
8	Mexico	Yucatan	Merida	30797.0	20141.2029...	6393.47822...	4262.31881...
1	Mexico	Guerrero	Acapulco	23593.0	17474.9897...	3670.80615...	2447.20410...
4	Mexico	Zacatecas	Camacho	23759.0	16844.4408...	4148.73549...	2765.82366...
5	Mexico	Jalisco	Guadalajara	24597.0	15011.5121...	5751.29269...	3834.19512...
12	Mexico	Zacatecas	Hidalgo	30584.0	21938.0009...	5187.59945...	3458.39963...
21	Mexico	DF	San Andres				
9	Mexico	DF	Mexico City	36509.0	22450.0263...	8435.38421...	5623.58947...
10	Mexico	Veracruz	Orizaba	34791.0	26354.0175...	5062.18949...	3374.79299...
18	Mexico	Zacatecas	Hidalgo	38382.0	30350.7148...	4818.77107...	3212.51405...
13	USA	OR	Salem	27694.0	18669.7842...	5414.52944...	3609.68629...
2	USA	WA	Bellingham	28206.0	22271.1505...	3560.90965...	2373.93977...
3	USA	WA	Bremerton	39696.0	24389.9213...	9183.64716...	6122.43144...
6	USA	CA	Beverly Hills	23688.0	15336.7531...	5010.74809...	3340.49873...
7	USA	CA	Los Angeles	23598.0	14210.3780...	5632.57318...	3755.04878...
0	USA	CA	Alameda				
11	USA	OR	Portland	20319.0	16231.7456...	2452.35258...	1634.90172...
24	USA	CA	San Diego				
14	USA	CA	San Francisco	22478.0	15320.9505...	4294.22967...	2862.81978...
16	USA	WA	Spokane	30268.0	22062.8673...	4923.07959...	3282.05306...
17	USA	WA	Tacoma	33858.0	22123.3454...	7040.79275...	4693.86183...
23	USA	WA	Yakima				
22	USA	WA	Walla Walla				
15	USA	WA	Seattle	21215.0	13305.4630...	4745.72215...	3163.81477...

d) Group On- Substring

1. Using the same Composite configuration as the previous illustration, open the Sort properties.

2. Select the City from the list of field names available in the combo box.

3. Select the sort order as "Descending" from the Combo box.

4. Select the Substring as the Group On option. Enter the value as 0,2 in the text box. Click the Ok button and Finish and then view the result.

The records are grouped based on the first 2 characters of the City field and sorted in descending order. So the Cities with names starting with "Ya" are grouped followed by names starting with "Wa" and so on.

e) Group On - Year

1. Instead of the Sales data source specified in the sort procedure given above, drag and drop the Employee data source and connect it to the Sort processor. Then open the Sort Properties
2. In the first screen of the Sort Wizard click the Add button. The Sort dialog box appears.
3. Select birth_date from the list of field names available in the combo box.
4. Select the sort order as "Descending" from the combo box.
5. Select the Year option from the combo box. Enter the value 2 in the Group Interval text box. The Sort dialog box appears as shown in the Figure 4.20, “Completed Sort Dialog”. Click the OK button. The sort column gets added to the Sort Wizard.

Figure 4.20. Completed Sort Dialog

6. Click the Finish button in the Sort Wizard and view the result.

The records are grouped based on the years and sorted in descending order. So records having birth_date as 1978-1979 are grouped together followed by 1976-1977 and so on.

Derivative Processor

The Derivative processor is used to derive one or more new columns through computations on existing fields present in the data source.

The Derivative processor is selected from the menu bar of the Designer Window and then placed on the diagram.

Properties

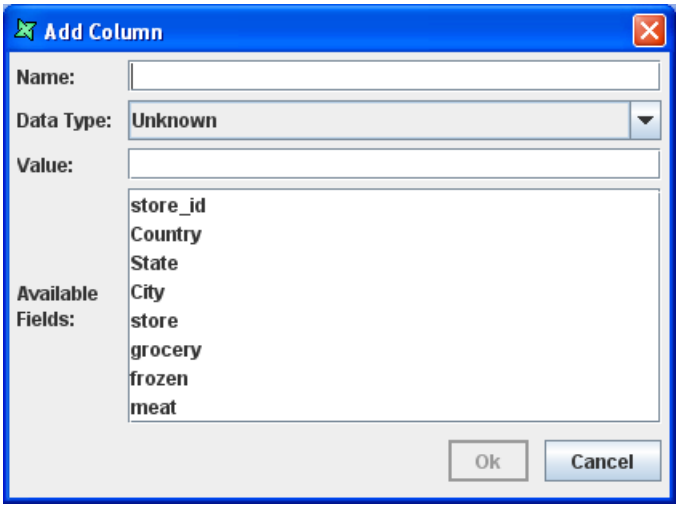
The editable properties are shown in Figure 4.21, “Derivative Wizard”.

Figure 4.21. Derivative Wizard

There are three tabs in the Derivative properties: Base, Derived and JavaScript.

- The Base tab will display all the fields present in the data source connected to the Derivative processor input.
- In the Derived tab there are 4 columns, Column, Name, Type and Value which contain input data required for deriving the new column.
- On clicking the Add button, the dialog box pops up as shown in Figure 4.22, “Add Column Dialog”. The following options are available.

Figure 4.22. Add Column Dialog



1. The name of the new column must be entered in the Name text box.
2. The data type of the derived column will be selected from the combo box.

3. The formula that will be used to derive the column or the JavaScript function name used to perform computations will be entered in the Value text box. On clicking the Ok button in the Add Column dialog box the derivative column is added to the Derivative Wizard.

- If more complex JavaScript functionality is needed then the code can be entered in the Derivative JavaScript tab. Note that any functions defined within this tab are only used within the scope of the Derivative - they can't be accessed elsewhere. If you need functions to be available throughout all uses of JavaScript, they should be defined on the Composite Diagram Script Tab instead.

- Highlighting row(s) in `Base` tab and right-clicking on it gives the option to add the selected row(s) to the `Derived` tab. Column name(s) containing spacing or symbols will be contained within square brackets with open and closed inverted commas, [""] automatically when moved over, but the Value of the column will have to be done manually.

Working with the Derivative processor

Here's how to derive two new columns from the Stores data source using the Derivative processor:

1. Add a JDBC data source, Stores, to the repository.

2. Add a Composite DataSource to the repository and open it.

3. Select Stores.ds, drag it into the Composite diagram and drop it.

4. Add a Derivative processor.

5. Connect Stores.ds to the Derivative processor and connect the output to Result. The designer window appears as shown in Figure 4.23, “Sample Derivative Flow”.

Figure 4.23. Sample Derivative Flow

Deriving a new column using a formula

Here's how to calculate the percentage of meat available in the stores:

1. After connecting the stores data source with the Derivative processor using the Derivative procedure given above invoke the Derivative Wizard by double clicking on the Derivative processor.

2. The fields available in the data source are listed in the Base tab.

3. Select the Derived tab. In this window click the Add Column button. The Add Column dialog box appears.

4. Enter the name as `meat_percentage` in the text box provided.

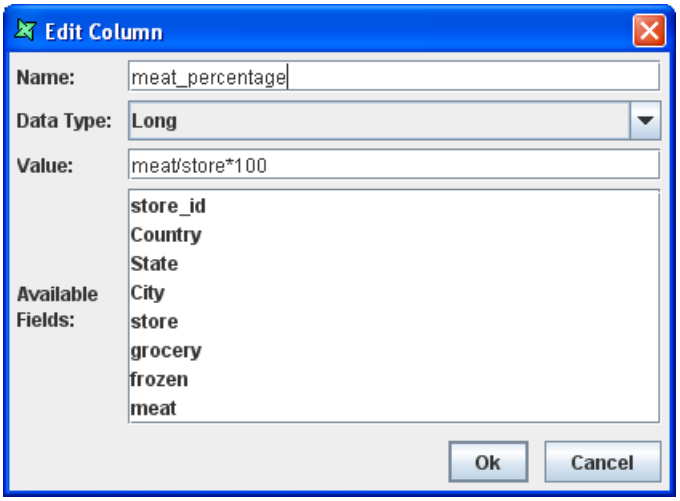
5. Select the Data type of the column as long.

6. Enter the formula

```
meat/store*100
```

in the value text box. The dialog box is shown in Figure 4.24, “Completed Add Column Dialog”.

Figure 4.24. Completed Add Column Dialog



7. Click the OK button and the column is added to the Derived tab window.

8. Click the Finish button and view the Result.

The output is shown in Figure 4.25, “Derived Result”. A new column meat_percentage has been added to the datasource, displaying the percentage of meat space available in the stores.

Figure 4.25. Derived Result

Designer | Script | Data

Showing 25 records | Count All Recor

store_id (Integer)	Country (String)	State (String)	City (String)	store (Double)	grocery (Double)	frozen (Double)	meat (Double)	meat_percentage (Long)
13	USA	OR	Salem	27694.0	18669.784...	5414.529...	3609.686...	13
2	USA	WA	Bellingham	28206.0	22271.150...	3560.909...	2373.939...	8
3	USA	WA	Bremerton	39696.0	24389.921...	9183.647...	6122.431...	15
6	USA	CA	Beverly Hills	23688.0	15336.753...	5010.748...	3340.498...	14
7	USA	CA	Los Angeles	23598.0	14210.378...	5632.573...	3755.048...	15
0	USA	CA	Alameda					
11	USA	OR	Portland	20319.0	16231.745...	2452.352...	1634.901...	8
24	USA	CA	San Diego					
14	USA	CA	San Francisco	22478.0	15320.950...	4294.229...	2862.819...	12
16	USA	WA	Spokane	30268.0	22062.867...	4923.079...	3282.053...	10
17	USA	WA	Tacoma	33858.0	22123.345...	7040.792...	4693.861...	13
23	USA	WA	Yakima					
22	USA	WA	Walla Walla					
15	USA	WA	Seattle	21215.0	13305.463...	4745.722...	3163.814...	14
8	Mexico	Yucatan	Merida	30797.0	20141.202...	6393.478...	4262.318...	13
1	Mexico	Guerrero	Acapulco	23593.0	17474.989...	3670.806...	2447.204...	10
4	Mexico	Zacatecas	Camacho	23759.0	16844.440...	4148.735...	2765.823...	11
5	Mexico	Jalisco	Guadalajara	24597.0	15011.512...	5751.292...	3834.195...	15
12	Mexico	Zacatecas	Hidalgo	30584.0	21938.000...	5187.599...	3458.399...	11
21	Mexico	DF	San Andres					
9	Mexico	DF	Mexico City	36509.0	22450.026...	8435.384...	5623.589...	15
10	Mexico	Veracruz	Orizaba	34791.0	26354.017...	5062.189...	3374.792...	9
18	Mexico	Zacatecas	Hidalgo	38382.0	30350.714...	4818.771...	3212.514...	8
19	Canada	BC	Vancouver	23112.0	16417.973...	4016.415...	2677.610...	11
20	Canada	BC	Victoria	34452.0	27463.353...	4193.187...	2795.458...	8

Deriving new columns using the various Date functions

Date manipulations usually require values such as 5 days ahead, 3 years ahead, 20 days before and 5 months before to be calculated.

1. Just like the previous illustration, connect Stores.ds to a Derivative processor.

2. After connecting, open the Derivative processor by double-clicking on the processor or right-clicking on it and select Properties.

3. The fields available in the data source are listed in the Base tab

4. Select the Derived tab. In this window, click the Add button.

5. Enter the name of the column as `Ahead_5_days`. Select Date as the Data Type. Enter

   ```
   offsetDays(first_opened_date,5);
   ```

   in the Value text box. This function will calculate 5 days ahead of the day in the given date. Click the OK button. The "Ahead_5_days" column is added to the Derivative Wizard.

6. Click the Add button, the Add Column dialog box pops up. Enter name of the column as `Ahead_3_years`. Select Date as the Data Type. Enter

   ```
   offsetYears(first_opened_date,3);
   ```

   in the Value text box. This function will calculate 3 years ahead of the year specified in the given date. Click the OK button. The "Ahead_3_years" column is added to the Derivative Wizard.

7. Click the Add button to add a new column. The Add Column dialog box pops up. Enter name of the column as `offset_20days_before`. Select Date as the Data Type. Enter

   ```
   offsetDays(first_opened_date,-20);
   ```

   in the Value text box. This function will calculate 20 days before the day specified in the given date. Click the OK button. The "offset_20days_before" column is added to the Derivative Wizard.

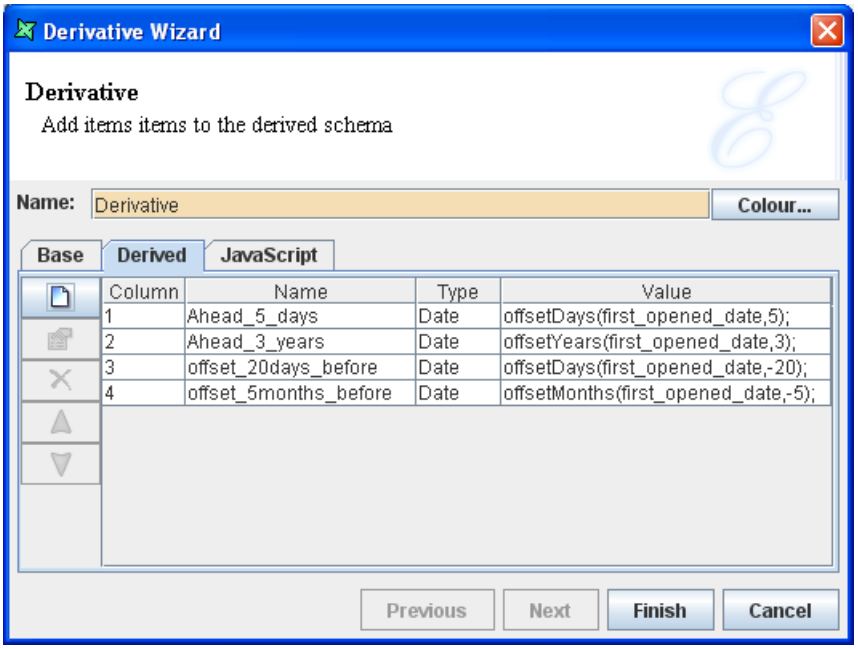
8. Click the Add button to add another new column. Enter name of the column as `offset_5months_before`. Select Date as the Data Type. Enter

   ```
   offsetMonths(first_opened_date,-5);
   ```

   in the Value text box. This function will calculate 5 months before the month specified in the given date. Click the OK button. The "offset_5months_before" column is added to the Derivative Wizard.

9. After entering the Column the Derivative Wizard appears as shown in Figure 4.26, “Completed Add Column Screen”. Click the Finish button in the Derivative Wizard.

Figure 4.26. Completed Add Column Screen



10. The output is shown in Figure 4.27, “Date Manipulations Result”. It can be seen that the offset dates are derived and displayed in new columns added to the data source.

Figure 4.27. Date Manipulations Result

Designer | Script | Data

first_opened_d (Timestamp)	Ahead_5_days (Date)	Ahead_3_years (Date)	offset_20days_before (Date)	offset_5months_before (Date)
1951-01-24 ...	1951-01-29	1954-01-24	1951-01-04	1950-08-24
1955-03-18 ...	1955-03-23	1958-03-18	1955-02-26	1954-10-18
1957-04-13 ...	1957-04-18	1960-04-13	1957-03-24	1956-11-13
1957-11-24 ...	1957-11-29	1960-11-24	1957-11-04	1957-06-24
1958-09-23 ...	1958-09-28	1961-09-23	1958-09-03	1958-04-23
1959-06-14 ...	1959-06-19	1962-06-14	1959-05-25	1959-01-14
1968-03-25 ...	1968-03-30	1971-03-25	1968-03-05	1967-10-25
1969-06-28 ...	1969-07-03	1972-06-28	1969-06-08	1969-01-28
1969-07-24 ...	1969-07-29	1972-07-24	1969-07-04	1969-02-24
1970-04-02 ...	1970-04-07	1973-04-02	1970-03-13	1969-11-02
1970-05-30 ...	1970-06-04	1973-05-30	1970-05-10	1969-12-30
1971-05-21 ...	1971-05-26	1974-05-21	1971-05-01	1970-12-21
1974-08-23 ...	1974-08-28	1977-08-23	1974-08-03	1974-03-23
1976-09-17 ...	1976-09-22	1979-09-17	1976-08-28	1976-04-17
1977-03-27 ...	1977-04-01	1980-03-27	1977-03-07	1976-10-27
1977-07-16 ...	1977-07-21	1980-07-16	1977-06-26	1977-02-16
1978-09-18 ...	1978-09-23	1981-09-18	1978-08-29	1978-04-18
1979-04-13 ...	1979-04-18	1982-04-13	1979-03-24	1978-11-13
1979-05-22 ...	1979-05-27	1982-05-22	1979-05-02	1978-12-22
1980-02-06 ...	1980-02-11	1983-02-06	1980-01-17	1979-09-06
1981-01-03 ...	1981-01-08	1984-01-03	1980-12-14	1980-08-03
1982-01-09 ...	1982-01-14	1985-01-09	1981-12-20	1981-08-09
1986-02-07 ...	1986-02-12	1989-02-07	1986-01-18	1985-09-07
1994-09-27 ...	1994-10-02	1997-09-27	1994-09-07	1994-04-27

Deriving a new column using the Java script function

This time, we'll calculate the percentage of Frozen area in the stores.

1. After connecting the Stores data source with the Derivative processor using the Derivative procedure given above open the Derivative Wizard properties.

2. The fields available in the data source are listed in the Base tab.

3. Select the JavaScript tab window and enter the following code:

```
function percent()
{
 percentage=frozen/store*100;
 return percentage;
}
```

4. Select the Derived tab window. Click the Add button the Add Column dialog box appears.

5. Enter the name of the derived field as frozen_percentage. Select long as the data type. Enter the function name

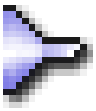
```
percent();
```

   in the value text box and click the OK button.

6. The column is added in the Derived window of the Derivative Wizard. Click the Finish button and view the Result.

A new column frozen_percentage has been added to the data source, displaying the percentage of Frozen area in the stores.

Filter Processor



Filter processor is a tool with which you can manipulate and group the database records to filter out those records which meet specific criteria.

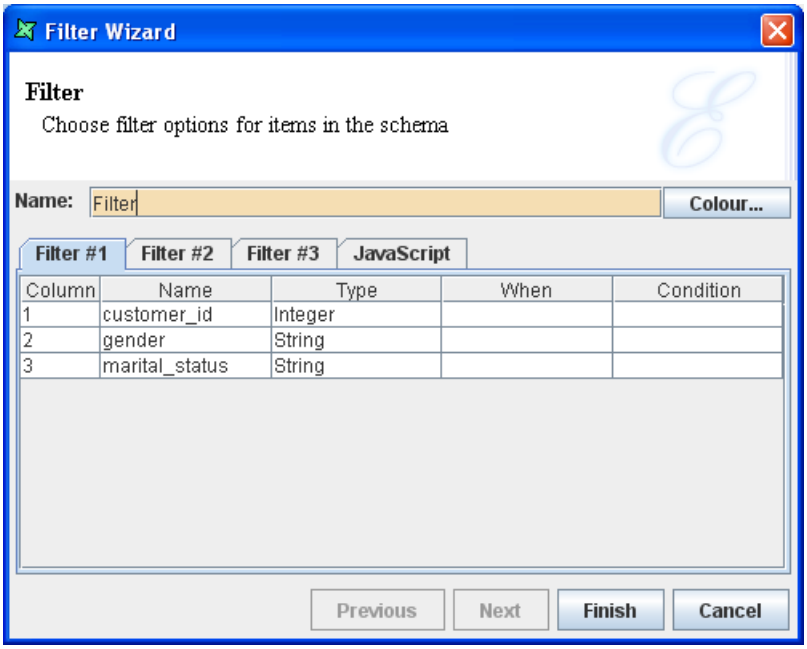
In the Data Designer, the filtering can be done by using the built-in functions or using the Javascript function for more complex filtering. Multiple levels of filtering are supported, up to three filter conditions can be set for each field within a single processor.

The Filter processor is selected from the menu bar of the Designer window and then placed in the designer window workspace.

Properties

The editable properties are shown in Figure 4.28, “Filter Wizard”.

Figure 4.28. Filter Wizard



There are four tabs in the Filter Wizard: Filter#1, Filter#2, Filter#3 and JavaScript.

All the Filter tabs have the following columns:

- Column - This column contains the row numbers.
- Name - This column contains the field names.
- Type - This column contains the data type of the records.
- When - A combo box in each field from which the filter condition can be selected.
- Condition - The filter value can be specified for the filter condition that has been set. You can also use dynamic parameters here to supply values at runtime.

Combining Filters

All criteria entered on a Filter tab must be true for the record to be passed through. For example, using /ElixirSamples/DataSource/FruitSales.ds as the input to the filter, setting Filter #1 to read

- CompanyName Equals A
- Fruit Equals Apple

will ensure that only those records where CompanyName=A **AND** Fruit=Apple will be passed through.

However, any records not matched by Filter #1, may still be matched by Filters #2 or #3. Within a tab the rule is AND. Across tabs, the rule is OR. So if we enter in Filter #2

- CompanyName Equals B

we will get those records where (CompanyName=A **AND** Fruit=Apple) **OR** CompanyName=B. Therefore we will also get all records where CompanyName is B, regardless of the value of Fruit.

Table 4.1. Filter Criteria

When	Condition	Remarks
• Equal/Not Equal • More Than/Not More Than • Less Than/Not Less Than	**Type:** Comparable types: String, Numeric, Date, Time, Timestamp **Example:** 10 1961-08-26	This condition uses a simple comparison of values, which must both be of comparable types. For example, strings can be compared with strings, but not with dates.
• Matches/Not Matches	**Type:** String **Example:** US|Mex	This condition allows a regular expression to be used. Use of operators like "|" for OR and "&&" for AND relationship. This example will match USA or Mexico strings.
• In Range/Not In Range	**Type:** Numeric, Date, Time, Timestamp with "~" as separator **Example:** 1998-01-01~1998-12-31	
• In DataSet/Not in DataSet	**Type:** Matching field type from another data source **Example:** See remarks	repository:/Sample/DataSource.ds:Field1 This retrieves the matching field from another data source for filtering values.
• Null/Not Null	Nothing to enter	Check whether if there is no content. Note that zero value is not considered null.
• JavaScript/Not JavaScript	Enter a script condition by specifying the matching type. **Example:** Field1==5|Field2=="US|Mex" where Field1 is numeric and Field2 is string.	This is useful for enter multiple conditions or complex conditions using standard JavaScript.

Working with Filters

1. Add a new Composite DataSource named `Filter`.
2. Add the JDBC data source for Customer and Stores by dragging and dropping them over the Composite diagram.
3. Place the Filter processor on the diagram.

4. Connect the Customer data source to the Filter processor and connect the output of the Filter to the Result. The designer window appears as shown in Figure 4.29, “Sample Filter Flow”.

Figure 4.29. Sample Filter Flow

Filtering records by setting the When condition in Filter tab window

To start, let's filter the records of married customers with customer_id less than 11:

1. Open the Filter Properties and in the row corresponding to customer_id field select `Less Than` from the When column. Enter the condition as 11.

2. In the Marital Status row select `Equals` from the When column and enter the condition as M. After setting the field properties the Wizard appears as shown in Figure 4.30, “Filter Result”.

Figure 4.30. Filter Result

Designer | Script | Data

[Result]

customer_id	gender	marital_status
1	F	M
3	F	M
4	M	M
7	F	M
8	M	M
9	M	M

3. Click Finish button and review the Result data. Notice that only the records of married customers with customer_id less than 11 are shown.

Filtering records using the JavaScript functions

Now let's find stores in Mexico with store_id less than 10. For a change we will use JavaScript, though this can be solved just as easily with the Filter tabs.

1. Disconnect the Customer datasource and connect the Stores datasource to the Filter input.

2. Open the Filter Properties and remove any existing filter criteria. Then select the JavaScript tab window.

3. Enter the following JavaScript syntax:

```
store_id<10 && Country=="Mexico";
```

   Note that because this expression is entered as a String, you can use dynamic parameters to customize the expression at runtime.

4. Click the Finish button and review the Result data. Notice that the records corresponding to Mexico with store_id<10 are fetched.

Note

If in the above syntax instead of "&&" the "||" symbols are specified then all the records corresponding to Mexico and all the records having store_id<10 are fetched.

Concat Processor

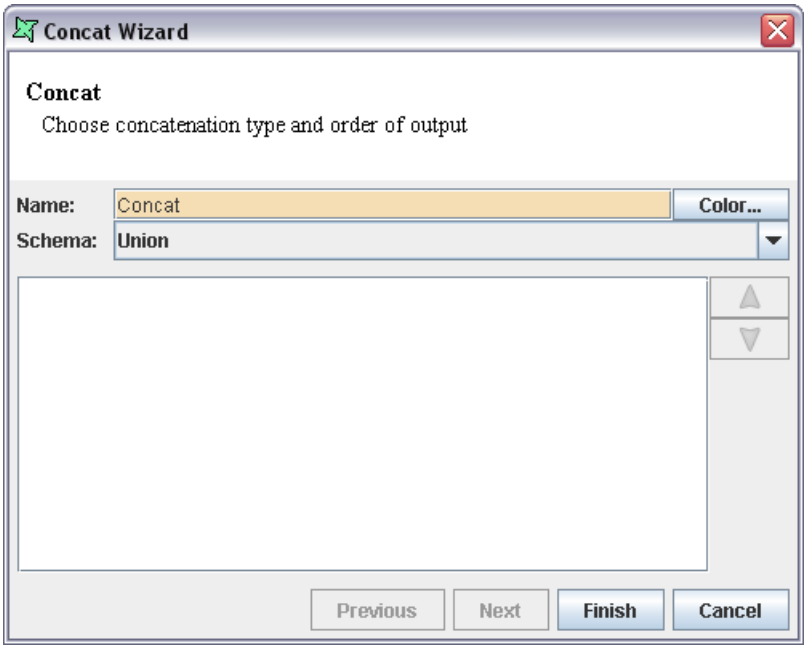
The Concat processor is used to concatenate the records of two or more data sources. If the schemas of the inputs are different, you can choose either the union of schemas (fields which don't exist in a particular input will be null) or the intersection of schemas (only fields which exist in all schemas will be retained).

The Concat processor is selected from the menu bar of the Designer window and then placed in the designer window workspace.

Properties

The editable properties are shown in Figure 4.31, “Concat Wizard”.

Figure 4.31. Concat Wizard



The Schema combo box contains the Union and Intersection options to choose which fields are passed through to the output.

The data sources that are connected to the Concat processor are listed in the window list box. The Up and Down arrows are used to move the selected data source up or down to change the order in which the records are concatenated.

Working with Concat Processor

As an illustration, let's concat the Stores and Sales data sources:

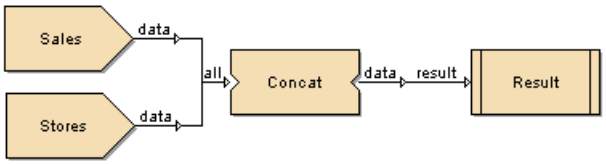
1. Add a Composite DataSource named `Concat`.

2. Select the Sales data source and drag and drop it onto the diagram. Repeat for the Stores data source.

3. Add a Concat processor to the diagram.

4. Connect the Stores data source and Sales data source to the Concat processor. Then connect the output of the Concat to the Result. The diagram appears as shown in Figure 4.32, “Sample Concat Flow”.

Figure 4.32. Sample Concat Flow



Combining all the fields of the data sources using the Union option

1. Open the Concat Properties and select the Union option from the Schema combo box.

2. The names of the data sources are listed in the window provided below the combo box.

3. Use the up and down arrows to order the inputs as desired - for example to ensure Stores records come before Sales.

4. Click Finish button in the Concat Wizard and review the output of Result. All the fields in the two data sources are combined together and displayed. Where no field data is available, the processor inserts nulls.

Combining the common field of the data sources using the Intersection option

Follow the procedure as outlined above, but now choose the Intersection option from the Properties dialog. Now when you review the Result output, you will see that only one field exists in both datasource schemas, store_id.

Parameter Processor

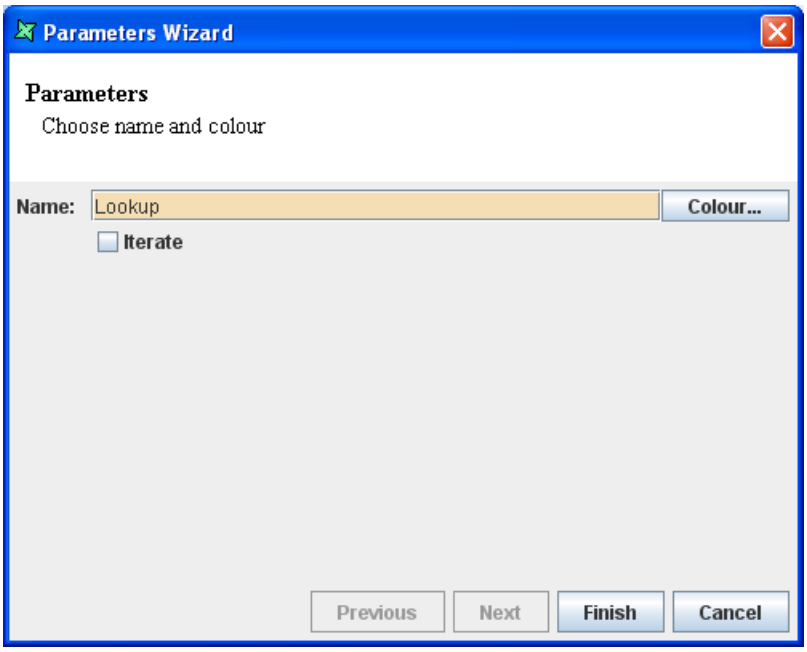
The Parameter processor retrieves parameters from one input and applies them to another flow. This allows flows to be developed and reused with different sets of parameters and for parameters to be maintained independent of the flows.

To understand the interaction of parameters, we need to understand the sequence of a flow. When we want to view data from a Result, the request is sent from the Result back up the flow to the source or sources. The source(s) then push the records back down the flow to the Result. The Parameter processor intercepts the original request as it travels up the flow and inserts the necessary parameter values. These are then available to processors further up the flow (towards the source). With this understanding we can see that Parameter processors are often used close to the Result-end of the flow, so that they can propogate parameters to all upstream processors.

Properties

The Parameter properties are shown in Figure 4.33, “Parameter Wizard”.

Figure 4.33. Parameter Wizard



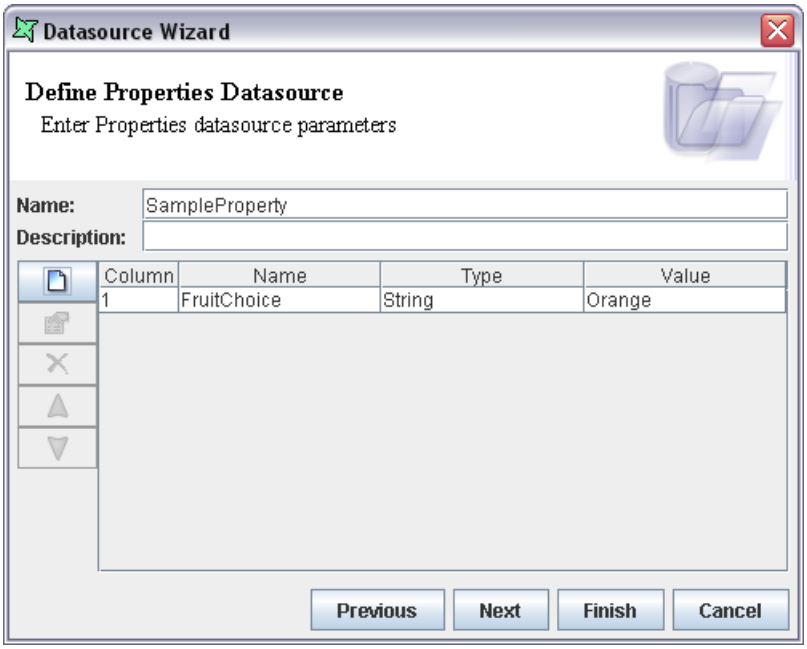
Working with Parameter processor

Let us filter records based on a parameter which is read from a different datasource:

1. Create a Properties DataSource named `SampleProperty`.

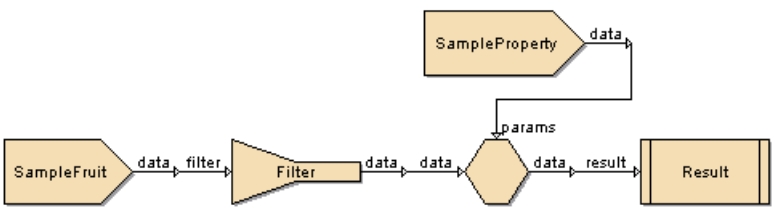
2. Click Add Column and in the dialog enter field Name as `FruitChoice`, Data Type as String and the Value as Orange. Click the OK button. The column is added to the DataSource. Similarly any number of fields can be added. After adding the fields the DataSource Wizard appears as shown in Figure 4.34, “Completed Properties DataSource”. Click the Finish button. (For more information on working with Properties DataSource refer to Chapter 8, *Properties DataSource*.)

Figure 4.34. Completed Properties DataSource



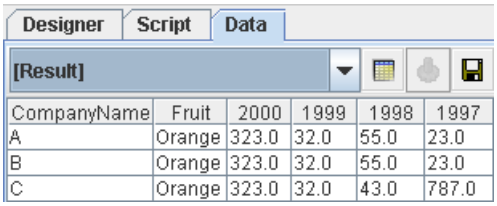
3. Add a Tabular DataSource `SampleFruit` which includes a column listing each fruit name.
4. Drag and drop `SampleFruit` onto the Composite diagram.
5. Place a Filter on the diagram and use a Flow to connect SampleFruit DataSource processor to the Filter processor.
6. Invoke the Filter Properties and in the Filter#1 tab select Matches from the When combo box in the Fruit row. Enter condition as ${FruitChoice} where ${FruitChoice} is the dynamic parameter we will get from the Properties datasource. For more information on dynamic parameters refer to Appendix A, *Dynamic Parameters*.
7. Click the Finish button.
8. Place a Parameter processor on the diagram.
9. Connect the Filter output to the data input of the parameter processor.
10. Drag and drop the `SampleProperty` onto the diagram.
11. Connect the SampleProperty data source with the `params` input of the Parameter processor.
12. Finally, connect the output of the Parameter processor to the Result. The diagram appears as shown in Figure 4.35, “Sample Parameter Flow”.

Figure 4.35. Sample Parameter Flow



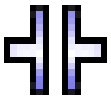
13. Select Result and choose View from the popup menu. The output is shown in Figure 4.36, “Parameter Result”. As the Value of FruitChoice in the property DataSource is Orange the records corresponding to Orange are fetched.

Figure 4.36. Parameter Result



CompanyName	Fruit	2000	1999	1998	1997
A	Orange	323.0	32.0	55.0	23.0
B	Orange	323.0	32.0	55.0	23.0
C	Orange	323.0	32.0	43.0	787.0

SubFlow Processor

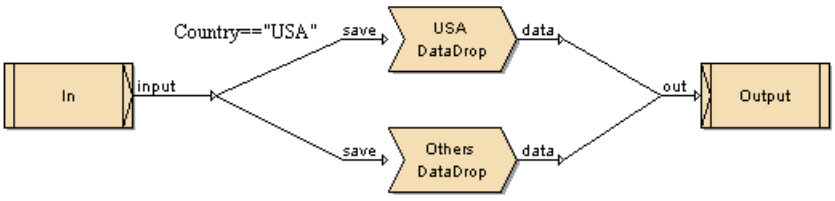


The SubFlow processor serves two purposes in Elixir Data Designer. Firstly it allows complex flows to be abstracted by defining modular sub-flows to aid comprehension of the diagram. In this role, a SubFlow acts like a subroutine call. In addition, SubFlow allows a single flow to be split into multiple branches based upon conditions. Each sub-flow may then perform a different sequence of actions, for example storing the records to disk, deriving values where data is missing etc. In this role, a SubFlow acts like a switch statement in a programming language.

The SubFlow processor is selected from the menu bar of the Designer window and then placed on the diagram.

The only editable properties of a SubFlow are name and colour. Double-click on the graphic, or choose View Diagram from the popup menu to open the nested diagram. You need to join the Input to the Output, with one or more flows, so that the records can be passed back to the parent flow. When acting as a subroutine, there is typically a single flow of processors between Input and Output. When acting as a switch there may be several flows in parallel. The flow in Figure 4.37, “SubFlow Sample” shows a switch-style flow.

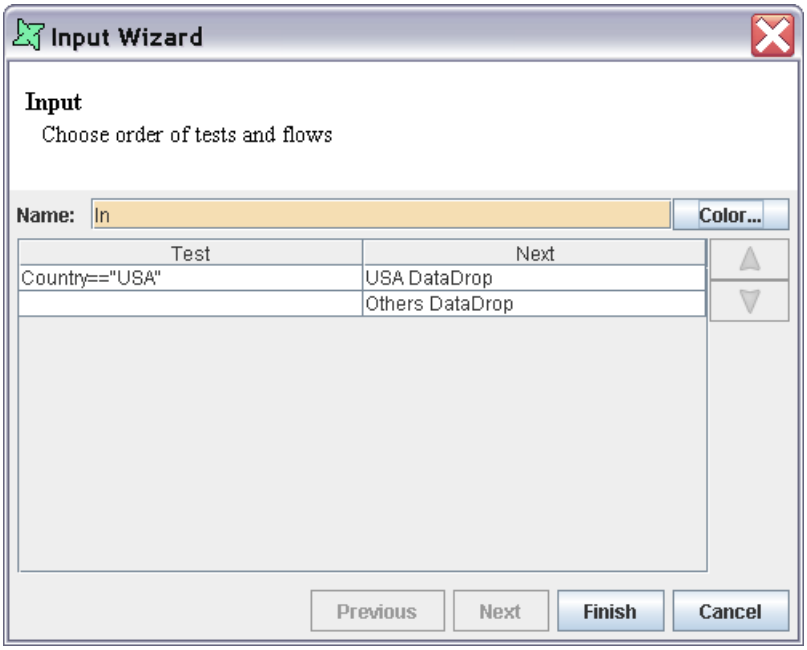
Figure 4.37. SubFlow Sample



To switch back to the parent diagram, choose the Close button from the subflow toolbar.

The Input graphic on the SubFlow diagram allows you to set a series of tests that are applied to choose between the different parallel flows. The wizard is shown in Figure 4.38, “Input Wizard”. This screenshot is taken from the SubFlow Sample above and shows how those records with Country value "USA" are passed to the USA DataDrop part of the flow and the rest (those with no test) are passed to the Others DataDrop. The Input tests proceed in order from top to bottom of the list. The first test that returns true indicates the record is passed to that flow. Subsequent tests are ignored. An empty test is considered true, and so all records that have not already tested true will pass to the first empty test flow.

Figure 4.38. Input Wizard



The test order can be controlled by moving the rows up and down in the table. If no test returns true, the record will be discarded. The tests will be shown on the diagram as labels attached to the appropriate flow. You can reposition the labels by clicking on them and adjusting the positions of the handles.

Note

The names of processors connected to the Input should be unique to make it easier to determine which flow corresponds to which test.

Processor

Elixir Data Designer provides a generic processor that allows certain specific data processing tasks to be executed. These tasks are designed to be easily customized in different versions of the tools, so you may find additional options available.

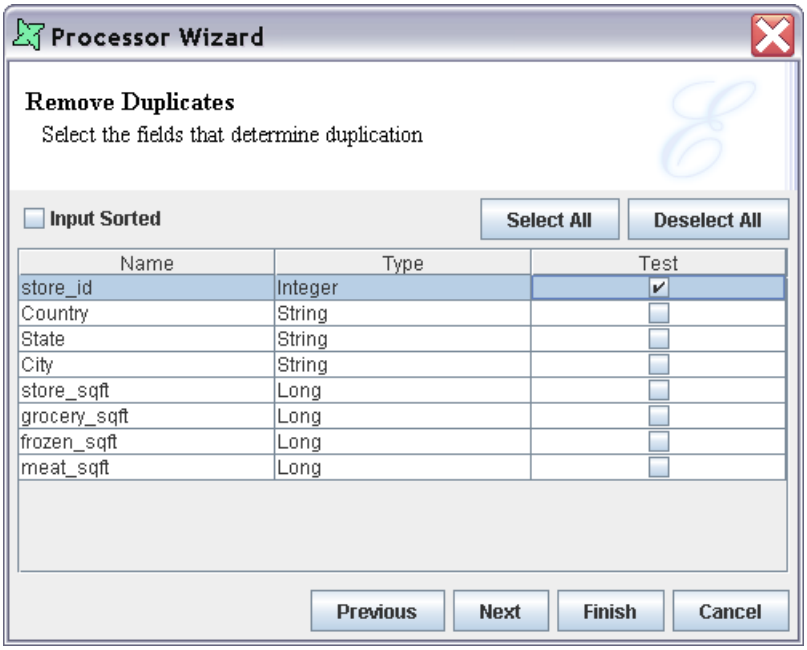
The standard processors are:

Cleansing

- **Remove Duplicates**

  This task removes duplicate records. A duplicate record is determined by a set of key fields being the same. The set may contain a single field, for example an identity card number, or a collection of fields, for example name and address. The fields are chosen by selecting a checkbox next to the appropriate items in the Test column of the schema table, as shown in Figure 4.39, “Remove Duplicates Processor”

Figure 4.39. Remove Duplicates Processor



Whenever subsequent records are identified as being duplicates of those already processed, the later records are always discarded. This means only the first record with a given set of key fields will be passed through. If the records are sorted such that duplicate records are adjacent, you should tick the Input Sorted checkbox so that the system can use an alternate algorithm to reduce memory usage. When the Input Sorted checkbox is selected, each record is only compared against the previous record, instead of all previous records, resulting in faster performance and reduced memory requirements.

General

- **Debug**

  This task allows the user to monitor the flow of records through the console without having to stop the flow. From there, user will be able to know where the problem lies and resolve the issue.

  Before being able to see the data flow through the console, user will need to edit *log-config.xml*. Look out for the following in the file and change `INFO` to `DEBUG`.

```
<logger name="com.elixirtech.data2">
  <level value="INFO" />
</logger>
```

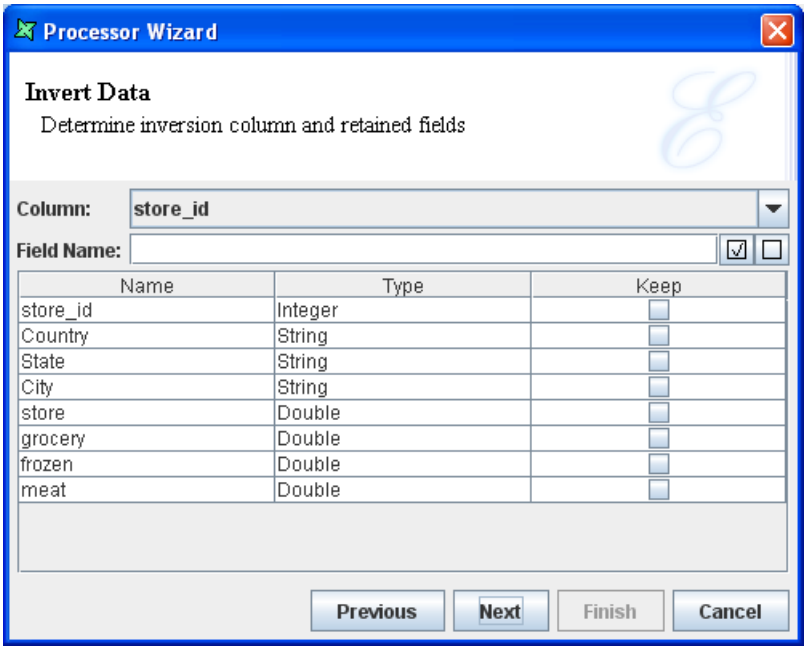
  This task is usually used after another processor which processes the data and it is at this stage where data gets messed up. Therefore, with the use of `Debug`, it will be easier to point out the source of problem.

- **Invert Data**

  This task inverts the data records so that the rows become the columns and vice-versa. In order to convert the rows to columns, one original column must provide the names of the columns in the new schema. This column should contain unique values to ensure the resulting schema doesn't have duplicate column names. Select the column that will provide the column names at the top of the wizard, as shown in Figure 4.40, “Invert Data Processor”. Conversely, you might want the old column names to be retained as row values. If so, enter a field name in the next text field, otherwise

leave it blank. Specifying a name will add a column in the output schema of type String, whose values will be equal to the Keep columns described below.

Figure 4.40. Invert Data Processor



The next step is to identify which of the old columns to keep - those that should be mapped into the new inverted structure. The resulting dataset will contain only as many records as are marked Keep. Because each row becomes a column when inverted, the system will need to determine the appropriate column type in the output schema. If the selected columns are all of the same type, then this type will be used. Alternatively, if the selected columns contain mixed types, then the String type will be used in the output schema to ensure all values can be represented.

As an example, assume a simple table like this:

```
A  B  C  D
1  2  3  4
5  6  7  8
```

Where A,B,C and D are the column names. If we choose InvertColumn=B, FieldName=Field, Keep=A,D then the output after Invert Data will be:

```
Field 2 6
  A   1 5
  D   4 8
```

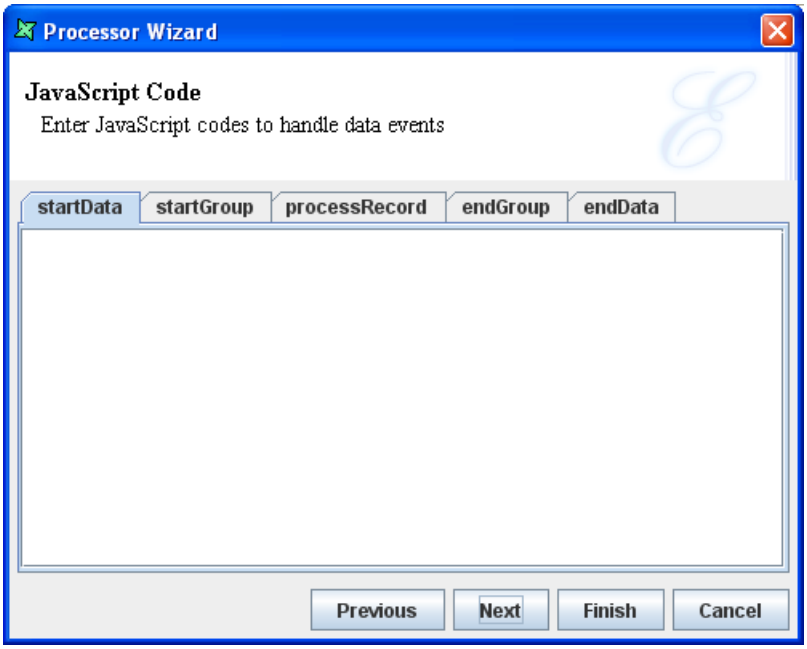
As you can see, because two columns were selected for keeping, there are two records in the output. The old column names are assigned to be the values of the new column called Field (the FieldName value), and the unique values of the B column (2 and 6) become the column names in the new schema.

- **Javascript**

  This task allows the user to perform any kind of record operation or schema re-arrangment which might be too complicated for regular flows. With this processor, a user can now save a lot of complex joins, filters and concatenations which is required previously.

  The scripts are to be entered in the text field in the respective tabs, as shown in Figure 4.41, “Javascript Processor”

Figure 4.41. Javascript Processor

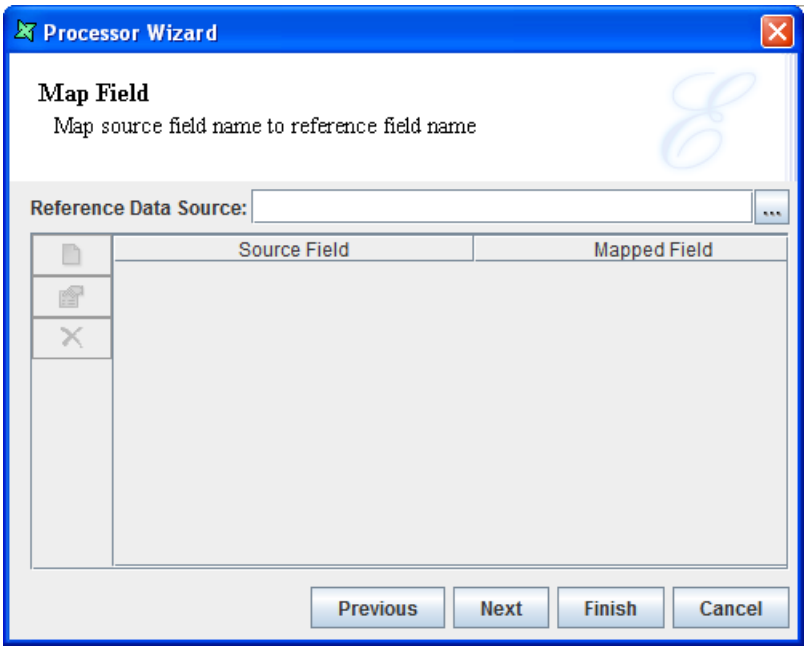


- **Map Field**

  Using `Map Field`, we can reference to another datasource and use the field(s) in that datasource and make our Composite datasource contain the field(s) in that selected datasource. The field name can be amended if desired.

  In the Map Field Wizard, choose a datasource from the repository to reference. Then add the fields to map.

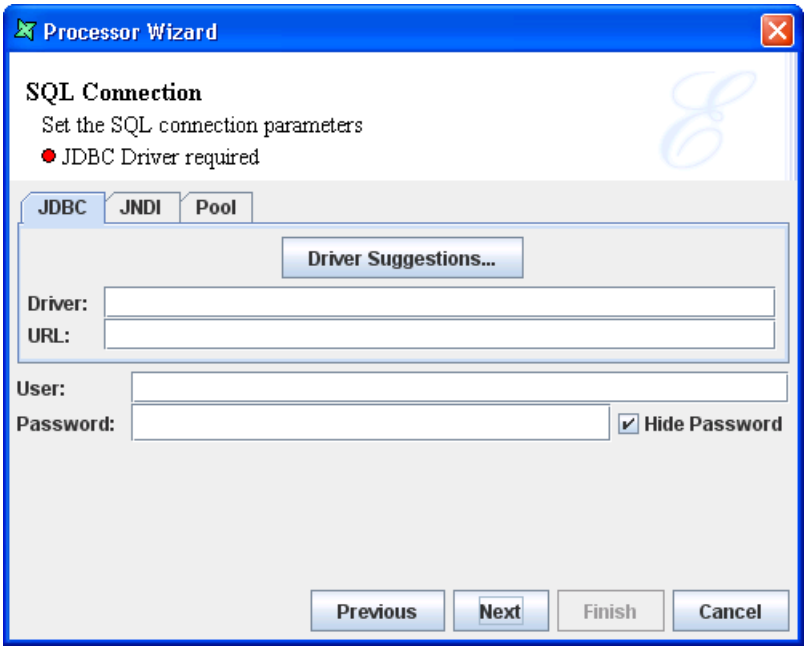
Figure 4.42. Map Field Wizard



- **SQL**

This task will allow the user to send any SQL commands to the database based on the standard flow events after a JDBC connection is set up on the second page of Processor Wizard, as seen in Figure 4.43, “SQL Processor Wizard”.

Figure 4.43. SQL Processor Wizard



The output of the SQL Processor is the same as the input and records passing through are not being modified as the user is only interacting with the database. With this, the user can load data in bulk into the database faster.

- **Trim**

  This task removes the leading and trailing spaces of string data.

DataDrop Processor

A DataDrop is a pass-through output mechanism used to generate the data that flows through it into different file formats.

A DataDrop is closely related to a DataStore. The only difference is that a DataDrop continues to provide records to subsequent processors, whereas a DataStore is a terminal processor.

A DataDrop is particularly useful when used in conjunction with a SubFlow construct, allowing different records to flow through different paths and be saved to different files.

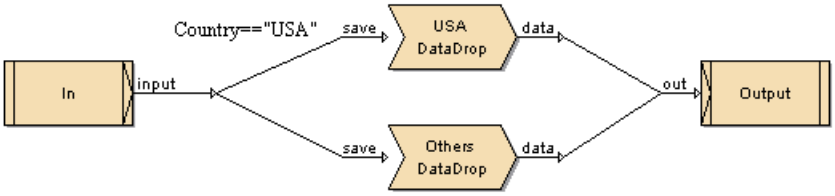
For more details on the features of DataDrop, review the DataStore section DataStore.

To explore the use of DataDrop, connect a SubFlow between a DataSource and a DataStore as shown in Figure 4.44, “Sample DataDrop Flow”.

Figure 4.44. Sample DataDrop Flow

Then, open the SubFlow (e.g. double-click on it) and draw the flow as shown in Figure 4.45, “Sample DataDrop Sub Flow”.

Figure 4.45. Sample DataDrop Sub Flow



In this case, the SubFlow input splits the records into two streams. The first stream has a condition attached: Country=="USA", the other stream is left blank, so it gets all the other records. Each of these streams passes through a different DataDrop, so the records end up in one of two files. All the USA stores are now separated.

DataDrop is particularly useful for storing records that fail data cleansing or data integrity checks for subsequent processing, or for debugging complex composite diagrams.

Note

- When trying to generate a Composite datasource with multiple Datadrop processors by right-clicking on the datasource in the Repository panel and selecting `Generate`, user is able to select the Datadrop processor to generate.
- From Repertoire 7.7 onwards, user can enter any symbol in the `Qualifier` field instead of selecting from the drop-down list.

Note

The Note graphic provides a simple way to annotate Composite diagrams. They carry no semantic meaning, so they can be attached as comments to describe the diagram for those viewing or maintaining it.

Cube Processor

Elixir Data Designer provides a Cube component which allows you to generate data results of the your choice by defining multi-level dimensions and multiple measures using the pre-defined functions like sum, average, count, max, min, etc.

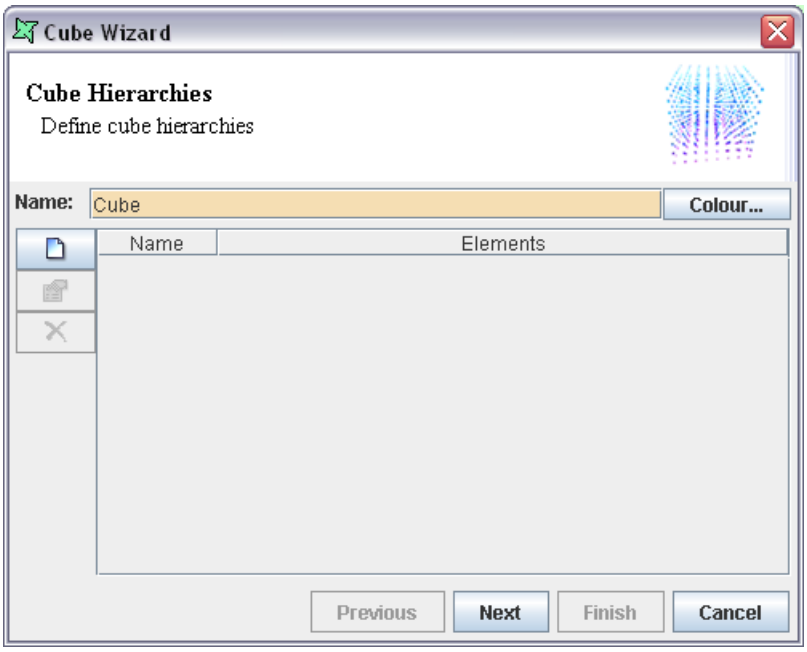
The Cube processor is selected from the menu bar of the Designer window and then placed in the designer window workspace.

The properties of the Cube processor are given below.

Properties

On opening the properties of a Cube processor, the dialog appears as shown in Figure 4.46, “Cube Wizard”.

Figure 4.46. Cube Wizard

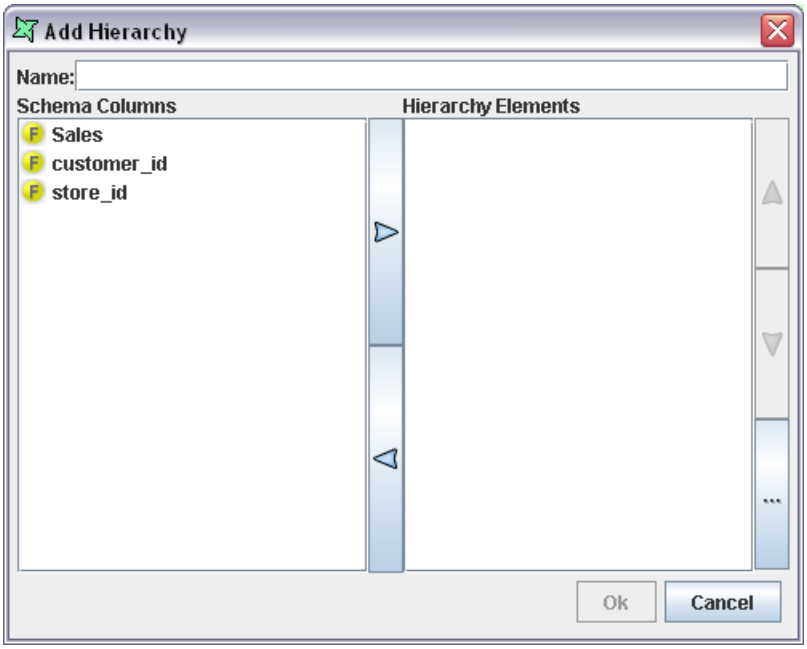


The Cube Hierarchies screen contains the Name and Elements column where the names of each hierarchy and the corresponding hierarchy elements will be listed.

There are three buttons on the screen namely Add, Edit and Remove which can be used to add, edit or delete a hierarchy.

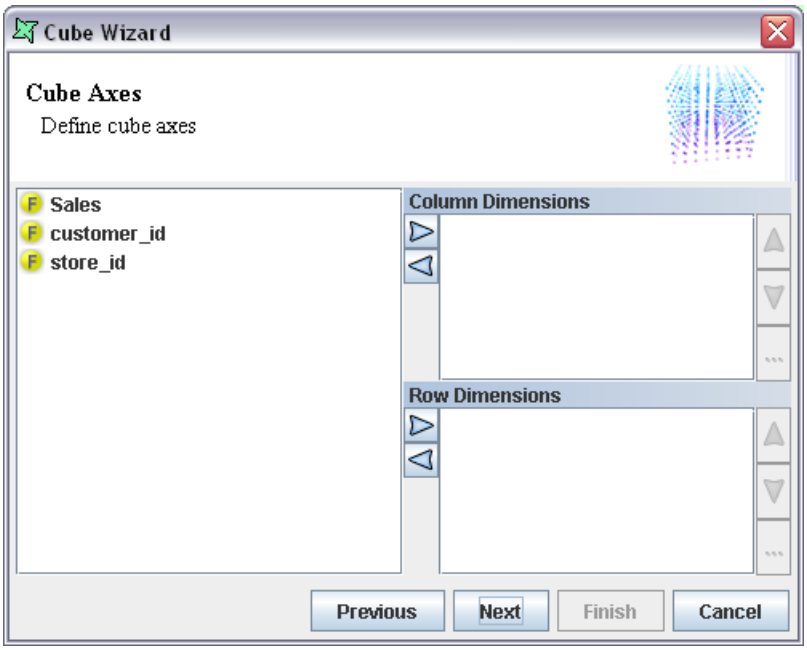
On clicking the Add button, the dialog box pops up as shown in Figure 4.47, “Add Hierarchy”, where the hierarchy elements can be added and a name can be assigned to the hierarchy. A hierarchy indicates a strict relationship between the hierarchy elements - each child can only belong to a single parent. For example, "Country/State/City" is a valid hierarchy because a City can't be in more than one State, and a State can't belong to more than one Country. "Year/Month" is not a valid hierarchy, because "January" can occur in many years.

Figure 4.47. Add Hierarchy



On clicking the Next button, in the Cube Wizard appears as shown in Figure 4.48, “Cube Axes Screen”. In this screen, the Column dimensions and Row dimensions can be added. The elements that are added in the Column dimension windows will be displayed as the column fields in the output table. The elements that are added in the Row dimensions window will be displayed as the row fields in the output table. The hierarchy column added in the first screen can be made use of in this screen as a row or column field. Use of a hierarchy is more efficient, as the system relies on the strict definition of each child only having one parent. However an incorrect use of hierarchy, as illustrated by "Year/Month" above will usually yield incorrect results.

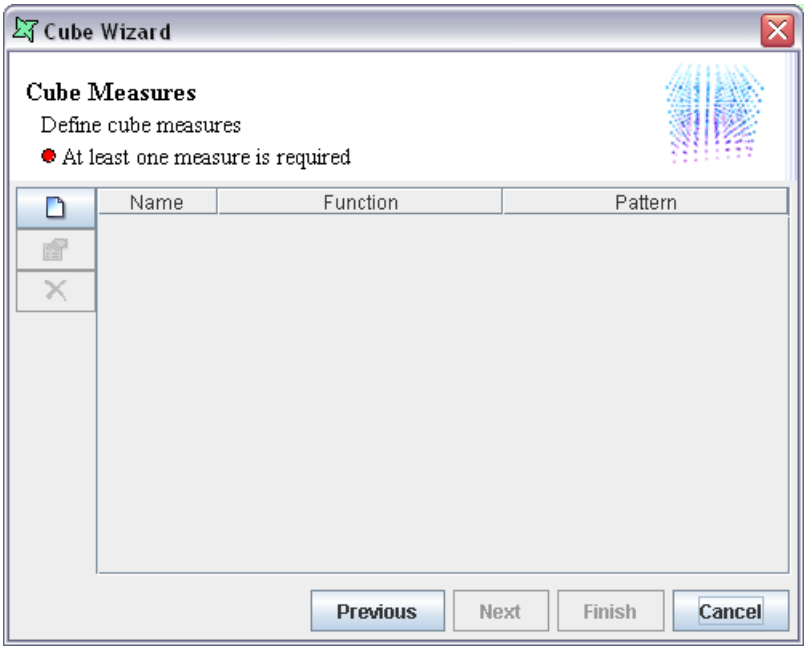
Figure 4.48. Cube Axes Screen



On clicking the Next button, the Cube Wizard appears as shown in Figure 4.49, “Cube Measures Screen”. There are two columns namely Name and Function. In this screen, the Pre-defined functions

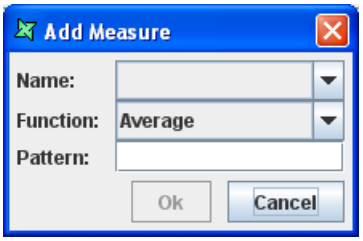
and the name of the fields using these functions will be listed. There are three buttons namely Add, Edit and Remove to add, edit or remove the cube measure columns.

Figure 4.49. Cube Measures Screen



On clicking the Add button, the dialog box pops up as shown in Figure 4.50, “Add Measure Dialog”. In this dialog box, the Field name, Function and Pattern can be selected. Each measure applies a function to a particular field, for example Average(Salary). The name of the measure is given by the values of the dimensions that form the output column. For example, Male/Average(Salary) and Female/Average(Salary) would be the column names for a Cube with a Gender column dimension. If there are two dimensions, eg. Country/Gender then you would get column names like Singapore/Male/Average(Salary).

Figure 4.50. Add Measure Dialog



Sometimes you might want to choose your own column names. This is where the Pattern field is useful. By default, if you leave the Pattern field blank, you will get column names as descibed above. If you enter a Pattern description, it will be used to generate the column names. There are a number of substitution parameters available. Assuming the default column name was US/Oregon/Married/Count(employeeid), here are some samples of the available substitutions:

- [blank] -> US/Oregon/Married/Count(employeeid) // backwards compatible
- ${*} -> US/Oregon/Married/Count(employeeid) // all, as before
- ${0} -> US // extract using index
- -${1}-${0}-${2}- -> -Oregon-US-Married- // another index eg.

- ${hierarchy} -> US/Oregon/Married // without the measure
- ${measure} -> Count(employeeid) // without the hierarchy
- ${hierarchy}/${measure} -> US/Oregon/Married/Count(employeeid) // all
- ${field} -> employeeid // just the field name
- ${fn} -> Count // just the measure fn
- ${fn} of ${field} -> Count of employeeid // varying formatting

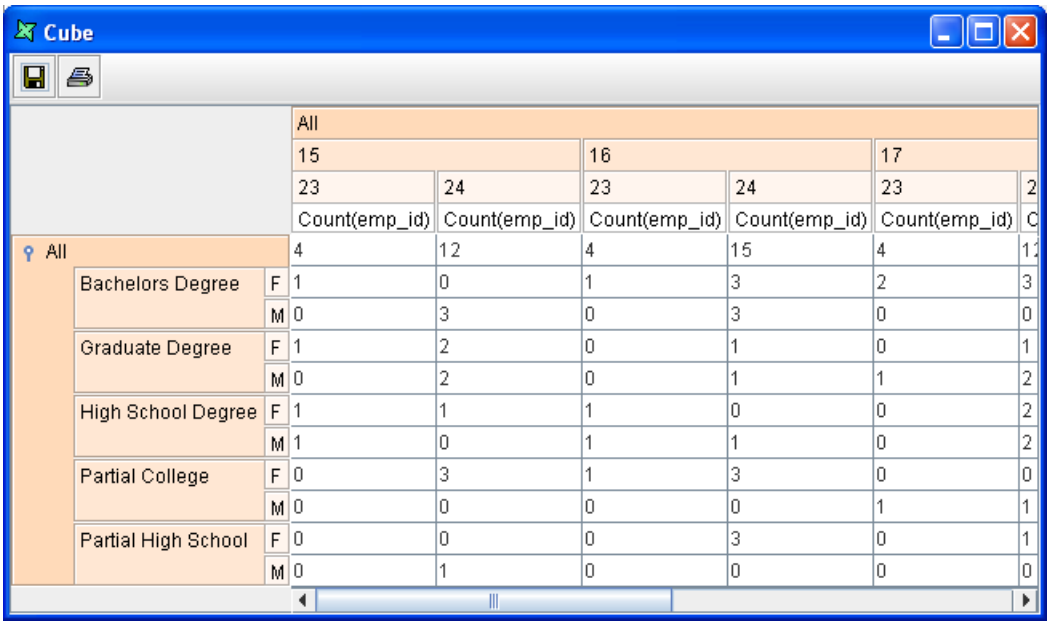
Note

It is important that any pattern you define should result in a unique column name for each column in the cube. This usually means that you should include the elements of the hierarchy and the measure somewhere in your pattern. Failure to ensure unique column names may prevent subsequent processor steps from accessing data from those fields with duplicate names.

The next wizard page provides a few options to tune the cube process and output:

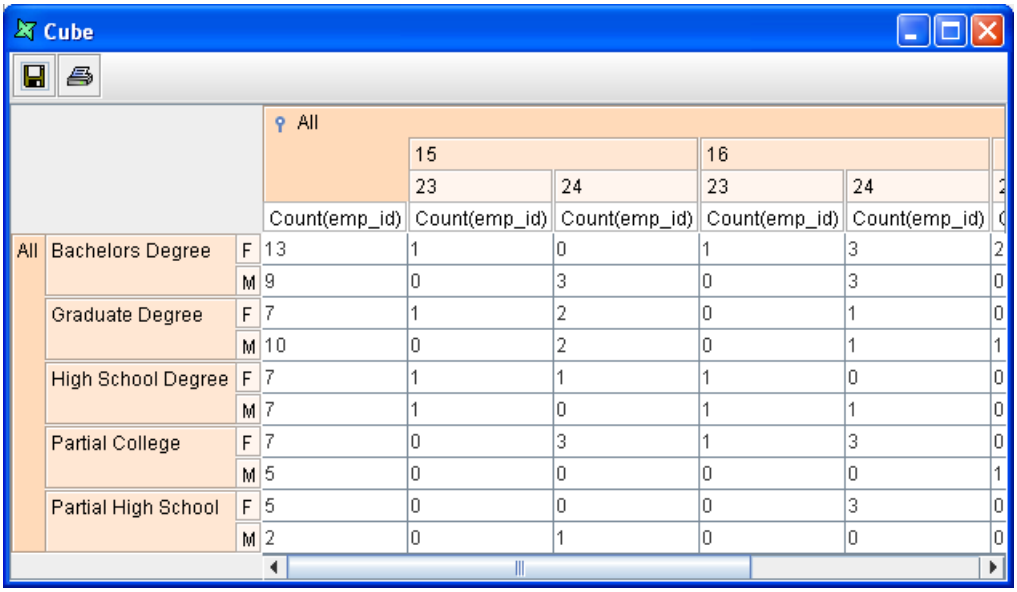
- **Collapse Empty Rows** This option will remove any rows from the cube that have no records contibuting to them. For example if Singapore/Male has no records, then only Singapore/Female would be shown. If empty records are not collapsed the output will be a permutation of non-hierarchical row dimensions. Hierarchies by definition (see strict hierarchies above) do not have empty rows.
- **Collapse Empty Columns** This option will remove any columns from the cube that have no records contibuting to them. Note that use of this option will alter the output schema - the columns indicate the fields each record will contain. Care should be taken when using this option as errors will result if subsequent steps depend on the presence of columns that are removed. The only benefit in removing empty columns is to improve presentation. For example, users of Elixir Report Cube Table can make use of this option to better utilize the available width of the report.
- **Show Row Grand Totals** When outputting records to subsequent processors, or Result, an additional row is inserted to represent the row total of the tree. Here's an example:

Figure 4.51. Cube - Show Row Grand Totals



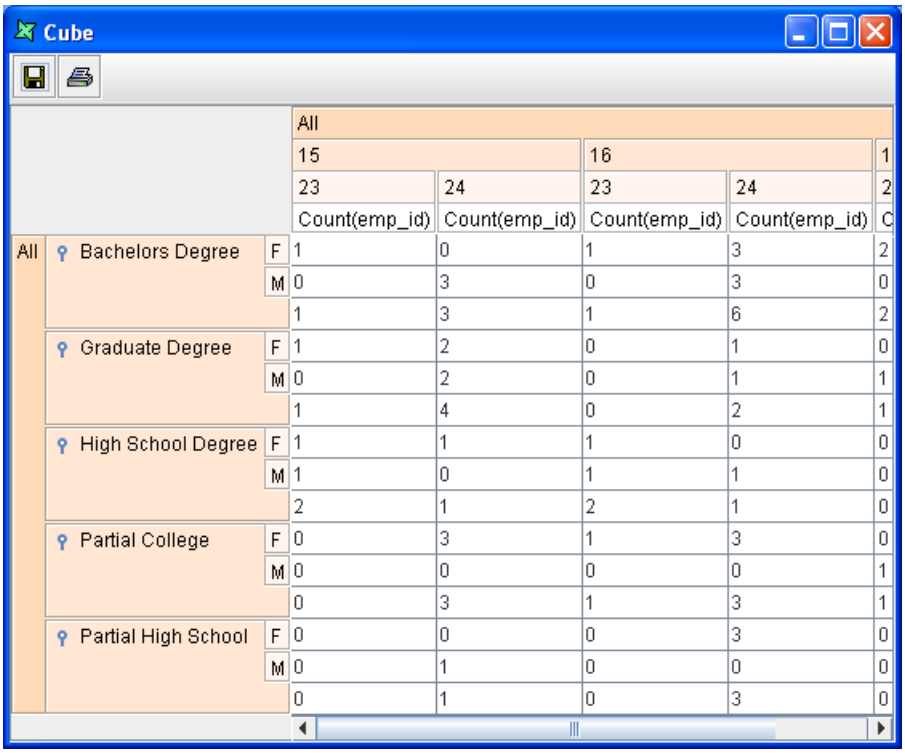
- **Show Column Grand Totals** As with Keep Row Totals above, this option inserts a total into the output. Each value at the outmost left of tree is the sum for the particular column.

Figure 4.52. Cube - Show Column Grand Totals



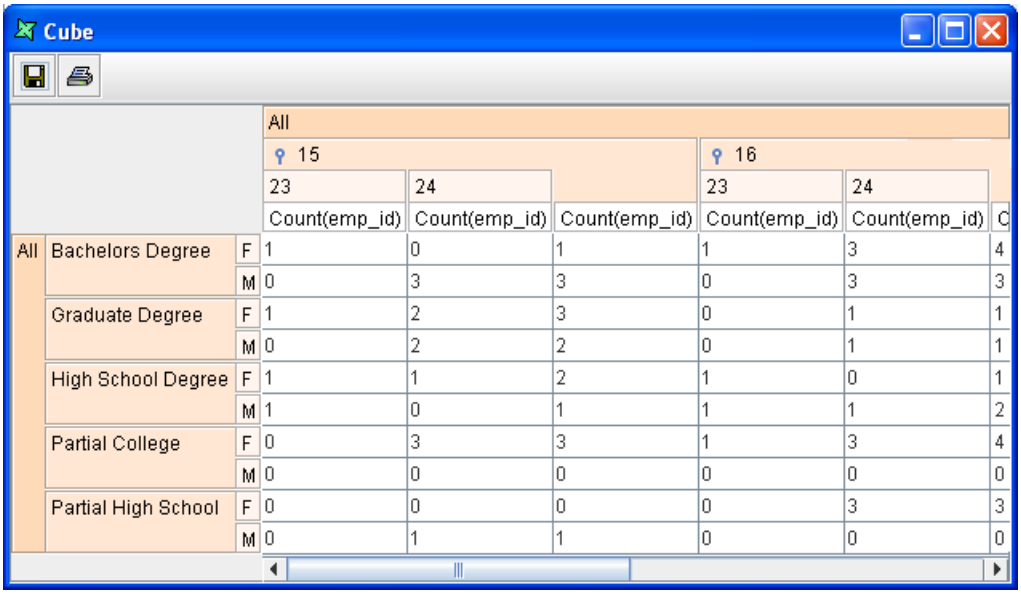
- **Row Totals After Details** Similar to *Show Row Grand Totals*, this option sums the total, but only after the row(s) of each group of Measure(s), as shown in Figure 4.53, “Cube - Row Totals After Details”.

Figure 4.53. Cube - Row Totals After Details



- **Column Totals After Details** This options works the same way as *Row Totals After Details*. The only difference is that this option creates a new column after each group of Measure(s). Each additional column is the sum of the values in that particular group.

Figure 4.54. Cube - Column Totals After Details

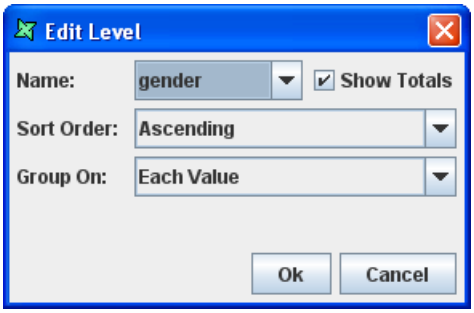


On clicking the Next button in the Cube Wizard the Infer Schema screen appears. The schema can be inferred by clicking the Infer Schema button. You should infer the schema after any changes to the dimensions or measures, including changing the measure patterns, because all these affect the column names.

Note

In order for *Row Totals After Details* and *Column Totals After Details* to display when generating the Cube, `Show Totals` must be checked, as shown in Figure 4.55, “Cube Axes - Edit Level”. As for *Show Row Grand Totals* and *Show Column Grand Totals*, it is not necessary to check the option `Show Totals`.

Figure 4.55. Cube Axes - Edit Level



Working with Cube Processor

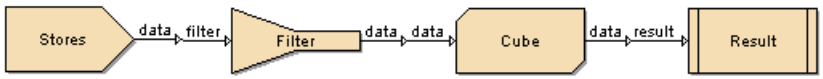
The averages of frozen and store for the stores in the different states of Mexico have to be compared.

1. Add the JDBC data source Stores.

2. Add a Composite DataSource named `Cube` and drag the Stores data source over the diagram.

3. Select a filter and place it in the designer. Next add a cube and connect the Stores to the filter, then to the cube and finally to Result. The diagram appears as shown in Figure 4.56, “Sample Cube Flow”.

Figure 4.56. Sample Cube Flow

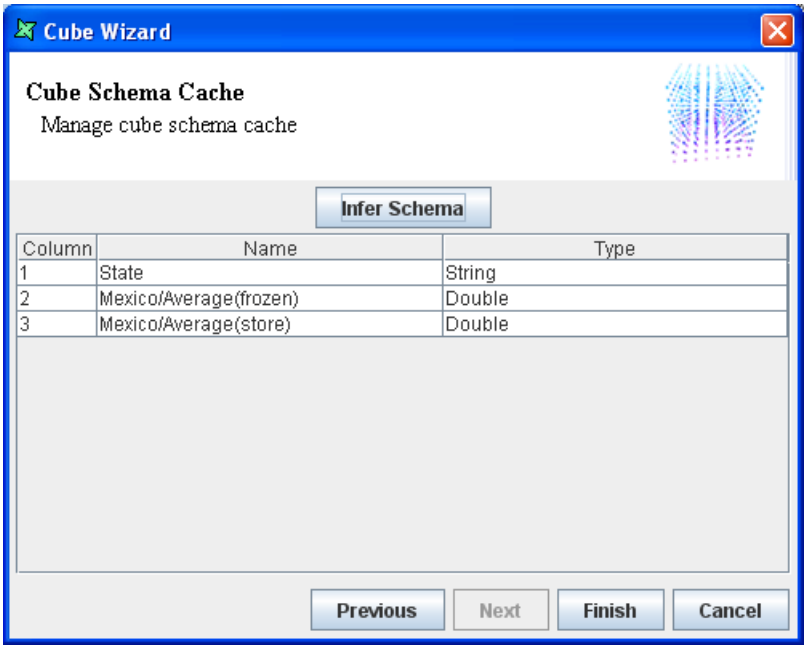


4. Open the filter properties and configure it to keep Country equals Mexico.

5. In the Cube Properties, go to the Cube Axes page, select Country as the cube column and State as the row. On the Cube Measures page, add a measure to Average the field frozen and add another measure to Average the store field.

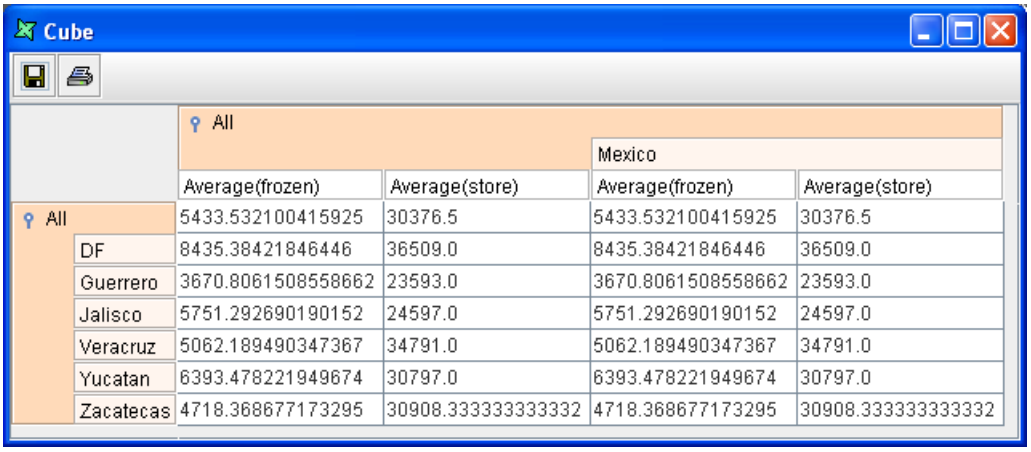
6. Click the Next button and infer the schema. The screen appears as shown in Figure 4.57, “Infer Schema Screen”. Click the Finish button.

Figure 4.57. Infer Schema Screen



7. From the Cube popup menu, choose View Cube. The output appears as shown in Figure 4.58, “Cube Result”. The averages of frozen and store for different states in Mexico are displayed.

Figure 4.58. Cube Result



DataStore Processor

A DataStore is an output mechanism used to generate the processed data into different file formats.

The different file types supported in Elixir Data Designer include CSV, Connection Pool, Custom Java DataStore, Excel, JDBC, JDBC JNDI, Text and XML.

From Repertoire 7.3 onwards, Clob and Blob data types can be written to the database using the DataStore processor.

The DataStore provides support for Command Line Invocation which allows Scheduling with any third party Scheduler such as Window Task Scheduler as well as an API for Application Integration.

If grouping is performed on a data source and the Excel DataStore is generated from it then the grouped data can be saved to multiple worksheets based on the grouping level.

The DataStore also supports XML post-processing with an XSLT transformation. An XSLT file contains a template that can transform the generated records into any textual output format.

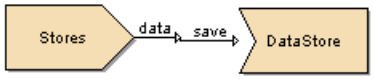
The DataStore is selected from the menu bar of the Designer window and then placed on the diagram.

The DataStore Wizard consists of two screens. In the first screen, the name of the DataStore must be entered in the text field. The DataStore type to be generated is selected from the list in the combo box. The output fields of the data source are displayed in the DataStore Wizard.

The URL and the other settings of the file to be generated will be entered on the second screen of the DataStore Wizard. The settings on the second screen vary with each output type.

To explore the use of DataStore, connect one directly to a DataSource and generate the different outputs. A sample is shown in Figure 4.59, “Sample Datastore Flow”. Of course, usually there will be a sequence of processors that act on the records as they pass through.

Figure 4.59. Sample Datastore Flow



Note

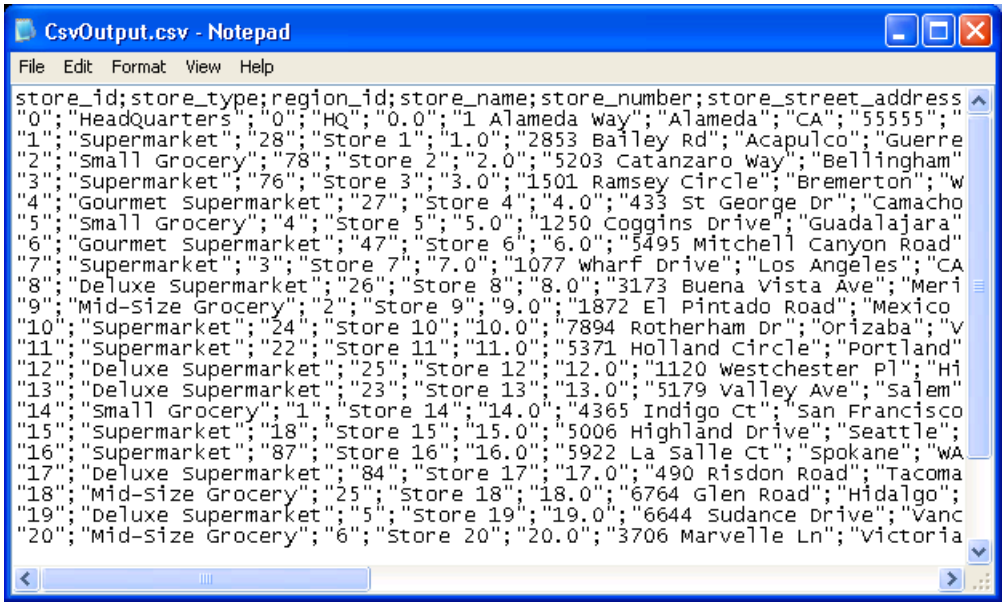
When trying to generate a Composite datasource with multiple Datastore processors by right-clicking on the datasource in the Repository panel and selecting `Generate`, user is able to select the Datastore processor to generate.

CSV File

This datastore writes comma separated values to a file. Each record will be output as a single line with the fields separated by a separator character. There is no trailing separator on the end of the line. In addition, the output can wrap the output values with qualifiers. These are usually single or double quotes which ensure that values which contain the separator aren't treated as multiple fields. For example if the field value is "Hello, World" and comma is used as a separator, the CSV will be malformed, unless a qualifier is used to delimit the field.

This datastore is not suitable for fields that contain embedded newline characters. Any such values should be fixed with earlier processors before the datastore. A sample output using semi-colon separators with double-quote qualifiers is shown in Figure 4.60, “CVS Output”.

Figure 4.60. CVS Output


CsvOutput.csv - Notepad
File Edit Format View Help
store_id;store_type;region_id;store_name;store_number;store_street_address
"0";"HeadQuarters";"0";"HQ";"0.0";"1 Alameda Way";"Alameda";"CA";"55555";"
"1";"Supermarket";"28";"Store 1";"1.0";"2853 Bailey Rd";"Acapulco";"Guerre
"2";"Small Grocery";"78";"Store 2";"2.0";"5203 Catanzaro Way";"Bellingham"
"3";"Supermarket";"76";"Store 3";"3.0";"1501 Ramsey Circle";"Bremerton";"W
"4";"Gourmet Supermarket";"27";"Store 4";"4.0";"433 St George Dr";"Camacho
"5";"Small Grocery";"4";"Store 5";"5.0";"1250 Coggins Drive";"Guadalajara"
"6";"Gourmet Supermarket";"47";"Store 6";"6.0";"5495 Mitchell Canyon Road"
"7";"Supermarket";"3";"Store 7";"7.0";"1077 Wharf Drive";"Los Angeles";"CA
"8";"Deluxe Supermarket";"26";"Store 8";"8.0";"3173 Buena Vista Ave";"Meri
"9";"Mid-Size Grocery";"2";"Store 9";"9.0";"1872 El Pintado Road";"Mexico
"10";"Supermarket";"24";"Store 10";"10.0";"7894 Rotherham Dr";"Orizaba";"V
"11";"Supermarket";"22";"Store 11";"11.0";"5371 Holland Circle";"Portland"
"12";"Deluxe Supermarket";"25";"Store 12";"12.0";"1120 Westchester Pl";"Hi
"13";"Deluxe Supermarket";"23";"Store 13";"13.0";"5179 Valley Ave";"Salem"
"14";"Small Grocery";"1";"Store 14";"14.0";"4365 Indigo Ct";"San Francisco
"15";"Supermarket";"18";"Store 15";"15.0";"5006 Highland Drive";"Seattle";
"16";"Supermarket";"87";"Store 16";"16.0";"5922 La Salle Ct";"Spokane";"WA
"17";"Deluxe Supermarket";"84";"Store 17";"17.0";"490 Risdon Road";"Tacoma
"18";"Mid-Size Grocery";"25";"Store 18";"18.0";"6764 Glen Road";"Hidalgo";
"19";"Deluxe Supermarket";"5";"Store 19";"19.0";"6644 Sudance Drive";"Vanc
"20";"Mid-Size Grocery";"6";"Store 20";"20.0";"3706 Marvelle Ln";"Victoria

Note

- The Append Data option allows data to be appended to the end of an existing CSV file. Obviously the same separator and qualifier options (and data schema) should be retained for each addition to the file.
- From Repertoire 7.7 onwards, user can enter any symbol in the `Qualifier` field instead of selecting from the drop-down list.

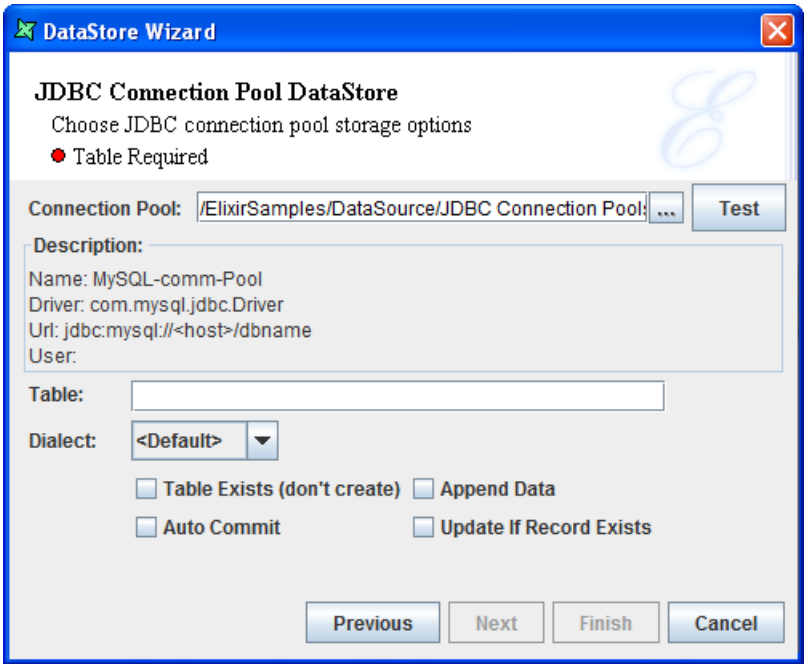
Connection Pool

This datastore writes records to the database that the connection pool is being connected to. If a valid URL to a connection pool is selected, the description of the connection pool will be displayed, as seen in Figure 4.61, “DataStore Wizard - Connection Pool”.

- *Table Exists(don't create)* : If this option is checked, the Table entered in the *Table* field will be updated accordingly. If not, a new table will be created. However, an error will occur if there is a table of the same name in the database.
- *Append Data* : When checked, data will be updated to the specified Table accordingly.
- *Commit Data* : Checking this option will enable data to be updated in the database whenever changes are made. If not, new records added will not be reflected in the database.
- *Update If Record Exists* : This option will only work if the destination table has primary key(s) defined. When checked, the system will check for the presence of record(s) based on the primary key. If found, it will perform an UPDATE. If not, it will perform an INSERT.

  This option is only available to JDBC, JDBC JNDI and Connection Pool.

Figure 4.61. DataStore Wizard - Connection Pool

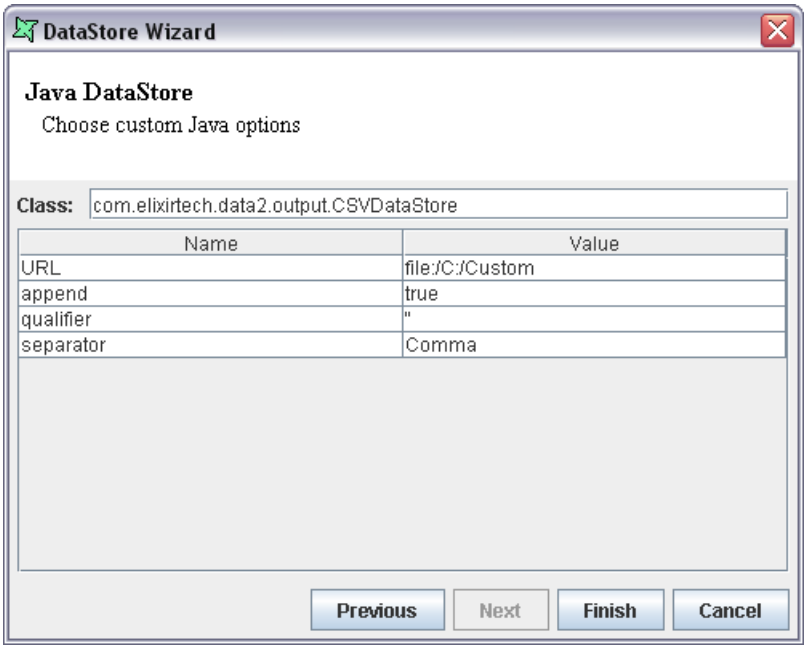


Custom Java DataStore

The Custom Java DataStore can be used to write data into any Java API, whether it be a proprietory protocol or even a mail or JMS queue. Actually, the built-in DataStore types are pre-defined instances of the Java DataStore. We can test this by using the CSV datastore as an example.

Open the DataStore Wizard and select Custom Java DataStore, then click Next. The Custom Java DataStore screen appears. Enter the class name as com.elixirtech.data2.output.CSVDataStore. The URL, append, qualifier and separator parameters appear in the table below. The CSVDataStore is written using standard JavaBean naming conventions, so the available accessors can be extracted automatically. Using CSV as an example, the DataStore screen appears as shown in fig Figure 4.62, “Custom Java Option Screen”.

Figure 4.62. Custom Java Option Screen



In order to implement your own DataStore you need to implement a DataListener interface. This API is described in the section called “Object DataSource API”. When you have coded and compiled your class, it should be placed in a jar in the /ext directory so that it can be loaded next time the tool is started. If you modify the jar you will need to restart Elixir Repertoire for the jar to be re-loaded. (If you are using the Remote tool, then the ext directory is on the server and you will need to restart the server). Now you can enter your class name in the Custom Java DataStore wizard and enter any values that you have exposed through get and set methods. Upon choosing Generate, your class will be invoked to process the records.

Excel File

This datastore writes records to an Excel (XLS) file. By default all records will be written to a single sheet. However, if the output records are grouped, the Sheet Group Level option can be used to force subsequent data to a new sheet. For example, if there are two levels of grouping and the level value is set to 1, then each start of a new level one group will start on a separate worksheet. Each sheet will have a header, if the Column Header check box is selected. An example with column headers is shown in Figure 4.63, “Excel Output”.

Figure 4.63. Excel Output

	C	D	E	F	G	H
1	State	City	store	grocery	frozen	meat
2	OR	Salem	27694	18669.78	5414.529	3609.686
3	WA	Bellingham	28206	22271.15	3560.91	2373.94
4	WA	Bremerton	39696	24389.92	9183.647	6122.431
5	CA	Beverly Hil	23688	15336.75	5010.748	3340.499
6	CA	Los Angele	23598	14210.38	5632.573	3755.049
7	CA	Alameda				
8	OR	Portland	20319	16231.75	2452.353	1634.902
9	CA	San Diego				
10	CA	San Franc	22478	15320.95	4294.23	2862.82
11	WA	Spokane	30268	22062.87	4923.08	3282.053
12	WA	Tacoma	33858	22123.35	7040.793	4693.862
13	WA	Yakima				
14	WA	Walla Wal				
15	WA	Seattle	21215	13305.46	4745.722	3163.815
16	Yucatan	Merida	30797	20141.2	6393.478	4262.319
17	Guerrero	Acapulco	23593	17474.99	3670.806	2447.204
18	Zacatecas	Camacho	23759	16844.44	4148.735	2765.824
19	Jalisco	Guadalajar	24597	15011.51	5751.293	3834.195
20	Zacatecas	Hidalgo	30584	21938	5187.599	3458.4
21	DF	San Andre				
22	DF	Mexico Cit	36509	22450.03	8435.384	5623.589
23	Veracruz	Orizaba	34791	26354.02	5062.189	3374.793
24	Zacatecas	Hidalgo	38382	30350.71	4818.771	3212.514
25	BC	Vancouver	23112	16417.97	4016.416	2677.611
26	BC	Victoria	34452	27463.35	4193.188	2795.458

Note

Different versions of Excel (and Excel compatible readers, like OpenOffice) have different limits for the maximum number of rows allowed in a single sheet. Elixir Data Designer limits the rows on one sheet to 31999.

JDBC

This datastore writes records to a JDBC database. The appropriate driver and database URL need to be entered, along with a user name and password (if required). The Table name may either by a literal table name, like `Sample`, or it may be inferred from a field in the record. If there is a field called CompanyName, with record values of A, B, C, etc. then using `${=CompanyName}` as the table name, will put the records into tables A, B and C. Each record could be appended to a different table.

Because table names may include spaces and other characters (particularly if they are read from field values), the datastore will wrap each string with double quotes "like this" when generating SQL codes. It may be useful to avoid this quoting if your table name also includes a schema component, for example myschema.table. This is because "myschema.table" is the name of a table in the default schema, whereas myschema.table is a table called "table" in the "myschema" schema. The latter is probably what you want, but we can't be sure. The solution is to explicitly quote the table name yourself. If quotes are already present (detected by an initial double quote mark), the datastore will not add any more. Therefore, if the table name is given as "myschema"."table" then the datastore will not introduce any extra quotes. This allows you to choose either interpretation of the table name explicitly. This discussion also applies for MySQL table names, except the back-tick (`) quote is used instead of double quotes throughout.

Note

The last page of the wizard for a JDBC datastore is for the user to specify the location to store records that failed to add, which are compiled into a datasource.

None

This datastore discards the records. This datastore is a useful trigger for priming data caches, or in conjunction with data drops. This is because only DataStores or Result can be triggered by name from the Elixir Runtime Engine. For example, if you want to ensure a cache is loaded, connect it to a None DataStore. Now you can generate this datastore any time you want the cache to be loaded.

Text File

This datastore writes expanded text templates to a file. By defining a template for StartData, StartGroup, ProcessRecord, EndGroup and EndData, you can control exactly how the text file is constructed. This is a flexible and fast alternative to using a Report Writer for producing text output.

A template is expanded by performing substitution of values using the familiar ${...} syntax. JavaScripts can be embedded too, by using ${=...} syntax. For example, to add the current timestamp to the output, you could use a template:

```
The current time is ${=java.util.Date();}
```

The StartData template is invoked first, so this is a good place for a header. If there are any groups in the output the StartGroup and EndGroup templates will be expanded at the appropriate point. Each individual record will be expanded using ProcessRecord.

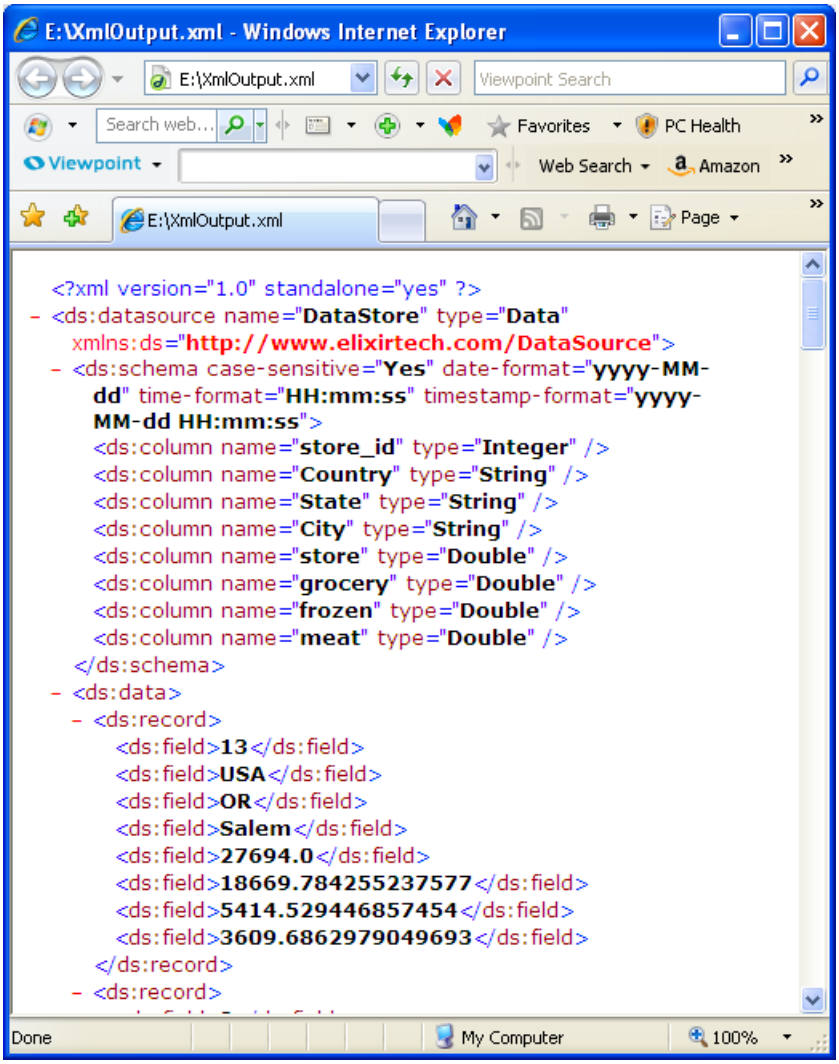
XML File

This datastore writes XML, or information transformed from XML to a file. By default, the records, along with the schema are written out in Tabular DataSource format - a simple XML structure that embeds the values directly into the data source itself, so that it has no other dependencies. You may take the XML file produced and use it with other tools, or by naming the output file with a .ds extension, you can use it as a data source. You can even write the file directly into the repository, by using a repository:/ URL so that Elixir tools can access it immediately.

By specifying the name of an XSLT transformation, the XML data output can be readily transformed into another XML structure, or any kind of text output. You could use XSLT to produce the same output as the CSV or Text DataStore, if they weren't already provided for you.

A Sample of the Tabular DataSource XML file is shown in Figure 4.64, “XML Output”.

Figure 4.64. XML Output



Note

If you need to send some sample data to Elixir, generating a Tabular DataSource using the XMLDataStore is ideal, because the data source file has no dependency on your database, repository or file system configuration.

Composite JavaScript

Composite DataSources have a JavaScript tab in addition to the Data tab and the diagram. Any scripts written in this JavaScript editor will be executed once when the Composite is about to begin processing. Therefore, this is a good place to define any functions and import any standard JavaScript libraries that you want to use throughout your Composite flow. All other JavaScript locations, for example in Derivatives and Filters etc. will execute once for every record that flows through them. It is inefficient to keep defining the same function over and over again, so move the functions themselves into the JavaScript tab and then just call the functions from the processors as needed.

For example, to filter a set of records so that only those with a SaleDate equal to today's date are retained, it would be useful to have an isToday(date) script. Here's one that can be put in the Composite JavaScript tab:

```
function isToday(date)
{
  var today = new java.util.Date();
  return today.year==date.year &&
 today.month==date.month &&
 today.day==date.day;
}
```

This function can now be called from inside a filter processor by choosing When type `JavaScript` with a Condition of `isToday(SaleDate)`. Where SaleDate is the name of the field holding the date value for testing.

Note

Fields of date type are actually java.util.Date instances. These are different from JavaScript Date instances - you need to make sure you are comparing like-with-like. You might notice that java.util.Date doesn't have a year attribute, but today.year still works because JavaScript will automatically invoke the appropriate get method for us.

Within a Composite processor script it is possible to refer to the fields of the current record by name, for example `SaleDate` in the example above. If the field name doesn't conform to JavaScript naming convention, for example if it starts with a digit, like `7Monkeys`, you can refer to it as `this["7Monkeys"]`. It isn't possible to refer to the fields in the Composite JavaScript tab, because this code is executed before the first record is read.

Within a Composite processor script it is also possible to refer to other processors by name (providing the name is unique). You can use this technique to lookup results from parallel flows or processors and incorporate them into your record sequence. The easiest way to describe this is through an example.

Suppose we have a data source called Countries with the following structure:

```
Code,EN,FR
UK,United Kingdom,Le Royaume-Uni
SG,Singapore,Singapour
US,United States,Les Etats-Unis
MY,Malaysia,La Malaisie
```

This could be from any kind of data source, but a Tabular DataSource is probably the simplest to create. Create a Composite DataSource and then drag and drop Countries onto the diagram. Now we repeat the process with another datasource called Source with this structure:

```
Name,Location
Jon,UK
Shih Hor,SG
```

Drop Source onto the diagram as well. Now create a new Derivative and connect it so the records will flow Source -> Derivative -> Result. Leave Countries unconnected. Now open the Derivative and create a new field called Location (which will overwrite the existing one) with the following value:

```
Countries.lookup("Code",Location,"EN");
```

View the result and you will see the Location changes from "UK" to "United Kingdom" because of the lookup.

The syntax of lookup is:

```
ProcessorName.lookup(field1,value,field2);
```

The function looks up the first record where `field1` contains `value` and returns the corresponding contents of `field2`. Null will be returned if it is not found. In our example, we lookup in the "Code"

column for a value "UK" and return the value of the "EN" column, which is "United Kingdom" (UK in English). If we wanted the French name, we would lookup("Code",Location,"FR"); and get back "Le Royaume-Uni".

Note

"Code" and "EN" are strings, but Location is a field - be sure not to quote it, it will be substituted with the Location value of the current record when the function is invoked.

You can now combine JavaScript lookups with other features, such as Dynamic Parameters to create:

```
Countries.lookup("Code",Location,"${Language#choice(EN,FR)#EN}");
```

which prompts the user for their chosen language (EN or FR) once and then proceeds to use the chosen language substitution for each record encountered.

Lookup works with any uniquely named processor, not just data sources. This means you could perform an operation on Countries before looking up the value). As with field names, you can use `this["7thProcessor"]` syntax to reference processors with non-conformant names.

Case Study

In this case study we will use the Sales, Customer and Stores tables of the Mondrian Database in conjunction with a variety of processors and generate the output into different file formats. The data manipulations we will illustrate are extraction, merging, filtering, derivation, caching and transformation.

Before we begin, you should ensure the Mondrian datasource is configured as described in the section called “Using the JDBC/ODBC bridge driver”.

Adding the DataSources

Launch Elixir Repertoire. Choose or create a new file system or folder for this case study and from the popup menu choose Add->Datasource. In the DataSource Wizard that appears select the JDBC DataSource. Click the Next button.

In the Define JDBC DataSource screen enter the DataSource name as Store. Select the JDBC/ODBC bridge(Sun JVM) as the driver specification. Enter the URL as jdbc:odbc:MondrianFoodMart. Click the Next button.

In the SQL window add the following query:

```
Select * from Store
```

Click the Next button and then click the Infer Schema button. Similarly, add the Sales data source with SQL:

```
Select * from Sales_Fact_1997
```

Again, infer the schema. Similarly, add a Customer data source. In the SQL window enter:

```
Select * from Customer
```

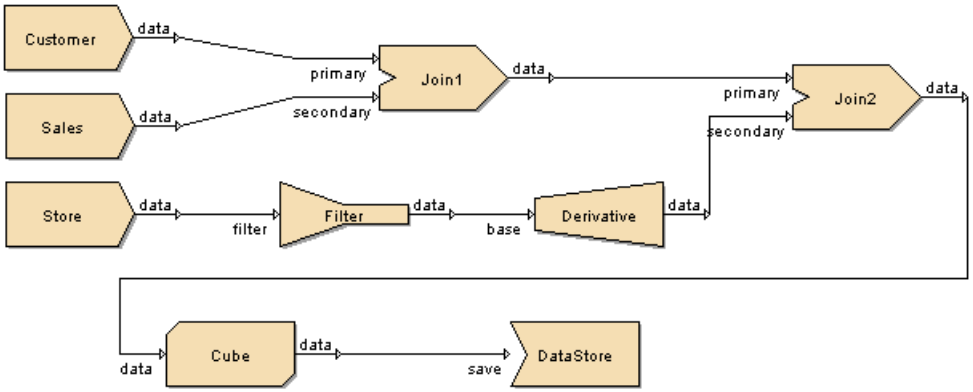
and again infer the schema.

Add a Composite DataSource named `Case Study`.

Creating a Composite DataSource

After adding the Composite DataSource, it will open automatically. We are going to create the diagram as shown in Figure 4.65, “Case Study Composite Diagram”. Select the Customer DataSource drag and place it on the diagram. Repeat the process for the Sales DataSource and then the Store DataSource.

Figure 4.65. Case Study Composite Diagram



Add the additional processors and connections as shown in Figure 4.65, “Case Study Composite Diagram”. In your version you will notice the link from Cube to DataStore is dashed because the tool cannot identify the schema for this flow until the cache has a schema inferred. With the diagram created, we can walk through the flow setting the various processor properties.

Join 1

- In the Options tab window enter If no matching secondary: Discard record. If multiple matching secondary: Repeat primary for every secondary.
- Change to the secondary tab and select the customer_id field in the "primary" column against the customer_id field of the secondary data source.

This configuration corresponds to an inner join, so records from the primary are only retained if a matching secondary record exists. In this case, only customers that have made purchases (have sales records) will be retained.

Filter

- In the Filter#1 select Equals from the When condition column corresponding to the store_country field. Enter condition as USA.

The output from this part of the flow will only contain records where the store_country field has the value USA.

Derivative

- Select the Derived tab and click the Add Column button to invoke the Add Column dialog box.
- Enter name as grocery_meat_sum. Select Data Type as Integer. Enter value as given below:

```
grocery+meat
```

- On clicking the Ok button the column is added to the wizard.

We've defined a new column grocery_meat_sum which contains the sum of grocery and meat fields.

Join 2

- In the Options tab window enter If no matching secondary: Discard record and If multiple matching secondary: Repeat primary for each secondary.

- Choose the Secondary tab and select the store_id field in the "primary" column against the store_id field of the secondary data source and click the Finish button.

Again we have used an inner join so records are only fetched based on a match with the secondary input. Because the secondary input filters out all countries except USA, the Join will discard all non-US customer sales from the primary datasource.

Cube

- In the Cube Hierarchies screen click the Add button.

  The Add Hierarchy dialog box pops up. Enter name as Location. Select store_country from the Schema column and click the > button. So the store_country is added to the Hierarchy Elements list. Similarly add the store_state and store_city to the Hierarchy Elements list. This creates a three-level hierarchy.

  On clicking the Ok button the hierarchy column is added to the Wizard. Click the Next button.

- On the Cube Axes screen, select gender and click > button to add the field to the Column Dimensions list box. Select Location and click > button to add the field to the Row Dimensions list box. Click the Next button.

- On the Cube Measures screen, click the Add button. The Add Measure dialog box pops up. Select store_sales and the function Sum, leave the Pattern blank. Clicking Ok to add the measure to the Cube.

  Similarly, add a measure for Average(store_sales) then Click the Next button.

- The next screen allows cube options to be configured. We will leave the options at their default values, so click next again and infer the schema. This operation may take a while as it is analysing hundreds of thousands of records. If you have previously executed the flow then the cache will have already been created and the inference will be faster.

You should notice that after we've inferred the schema that the flow connector from the Cube to the DataStore has changed from a dashed line to a solid line, indicating that a schema is now defined for this link. We can test the process so far by selecting the Cube processor and choosing View Cube from the popup menu. The output is displayed as shown in Figure 4.66, “View Cube Output”.

The sum and average of sales for the male and female customers for the different cities belonging to specific States of USA are displayed.

Figure 4.66. View Cube Output

			All			
			F		M	
			Sum(store_sales)	Average(store_sales)	Sum(store_sales)	Average(store_sales)
All	USA		280226.21000000346	6.542602554224825	285011.92000000546	6.476660455392571
		CA	79050.7900000001	6.592510215995338	80117.04999999964	6.4345875833266115
		Beverly Hills	23288.109999999946	6.797463514302378	22462.129999999986	6.627952198288576
		Los Angeles	25908.679999999942	6.7051449275362165	28636.600000000035	6.593737048123425
		San Diego	27588.159999999956	6.815256916996036	26842.979999999978	6.632809488510002
		San Francisco	2265.8399999999997	3.469892802450229	2175.3399999999965	3.23711309523809
		OR	69825.44000000018	6.626690708930452	72451.62999999976	6.542498645475868
		Portland	26408.839999999927	6.673955016426567	28649.949999999957	6.651950313443222
		Salem	43416.599999999955	6.598267477203641	43801.679999999935	6.472835820895512
		WA	131349.98000000056	6.469486282815375	132443.24000000066	6.466639324251778
		Bellingham	2322.340000000001	3.4354142011834337	2416.89	3.4330823863636364
		Bremerton	24979.130000000005	6.655776711963764	27917.170000000016	6.771081736599568
		Seattle	28369.209999999926	6.595956754243182	24274.859999999957	6.6415485636114795
		Spokane	25486.37999999994	6.712241243086631	24148.080000000016	6.707800000000004
		Tacoma	37286.79999999995	6.73289996388587	37557.15999999998	6.651994332270631
		Walla Walla	2184.610000000001	3.4081279251170065	2521.3600000000015	3.612263610315188
		Yakima	10721.509999999997	6.713531621790856	13607.720000000016	6.621761557177623

DataStore

You can choose a few different kinds of output for the records we have processed, for this walkthrough we have chosen XML, MySQL and Oracle.

XML: Invoke the DataStore Wizard and set the following properties:

- Select XML file from the Type combo box. Click the Next button.
- Enter the URL as file:/C:/Output.xml (choose an appropriate location for your operating system) and click the Finish button.

Select the DataStore, and select Generate from the popup menu. The XML file will be generated and saved in the specified location.

MySQL: Before generating the MySQL JDBC DataStore, the MySQL driver file must be copied to the Elixir Repertoire ext folder. The tool must be launched once the jar is in place, as the jar is only loaded at startup. (Note if using Elixir Repertoire Remote, then the ext folder is on the server.)

Invoke the DataStore Wizard and set the following properties:

- Select JDBC from the Type combo box. Click the Next button.
- Select MySQL in the suggestions combo box.
- On selecting the Driver Suggestion, the Driver class name and the URL are automatically displayed in the corresponding text boxes.
- The URL is entered as jdbc:mysql://localhost:3306/test. Where test is the dbname, localhost can be replaced by the IP address of MySQL server if it isn't running on the same machine and 3306 is the port number.
- Enter DataOutput as the table name.

Select the DataStore and choose Generate from the popup menu. The MySQL JDBC DataStore is generated and saved in the specified location.

Oracle: Before generating the Oracle JDBC DataStore, the Oracle driver file must be copied to the ext folder (on the client for the Designer, or on the server for the Remote). This ensures the Oracle driver is loaded into the class path when the tool is launched.

Invoke the DataStore Wizard and set the following properties:

- Select JDBC from the Type combo box. Click the Next button.
- Select Oracle in the suggestions combo box.
- On selecting the Driver Suggestion, the Driver class name and the URL are automatically displayed in the corresponding text boxes.
- The URL as jdbc:oracle:thin:@localhost:1521:ELX. Where localhost should be replaced by the IP address of the Oracle server if it isn't running on the same machine. The number 1521 is the port number and ELX is the database name.
- Enter DataOutput as the table name.

Select the DataStore and choose Generate from the popup menu. The Oracle JDBC DataStore is generated and saved in the specific location.

Chapter 5
Excel DataSource

Overview

Excel is a spreadsheet application developed by Microsoft. By adding an Excel DataSource you can access the data directly from the Excel file. From Repertoire 7.4 onwards, XSLX format (Excel 2007 format) will be supported.

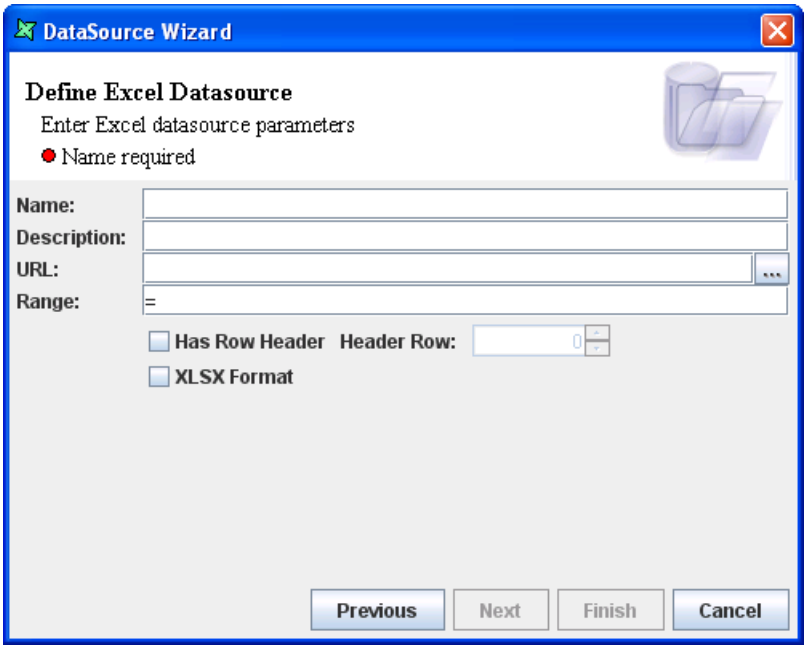
A range of data can be loaded, including support for both single and multiple sheet (3D) ranges.

An Excel DataSource can be created simply by right-clicking on an Excel file on the repository and select “Define as an Excel DataSource”.

Excel DataSource Wizard

A wizard will guide you through the process of creating a new Excel DataSource as shown in Figure 5.1, “Excel DataSource Wizard”.

Figure 5.1. Excel DataSource Wizard

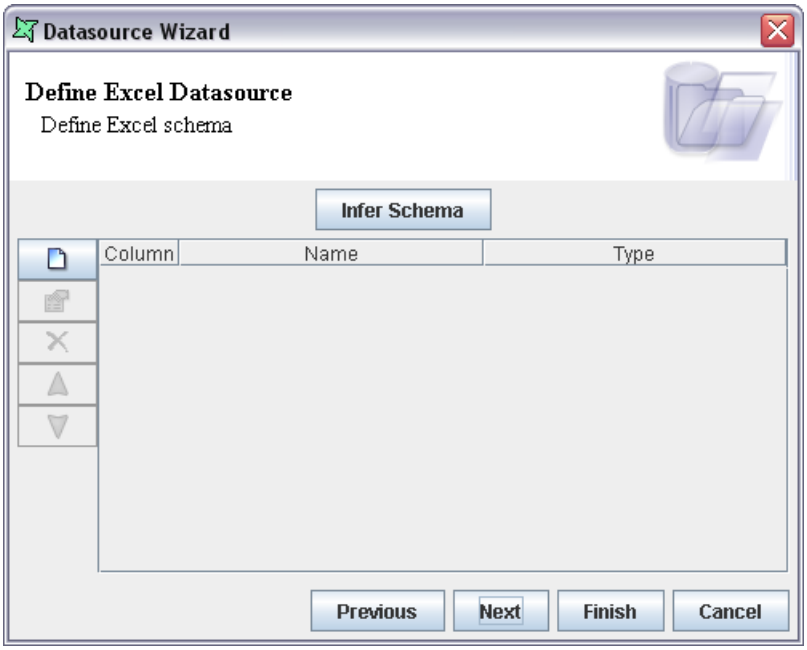


The Excel DataSource Wizard contains the Name, Description, URL and the Range fields.

- The DataSource Name is entered in the Name text box.
- Any extra description for the data source can be entered in the Description field.
- The URL of the file is specified in the text field.

- The data range must be specified in the Range field. The records will be fetched based on the specified range by inferring the schema in the next screen.

Figure 5.2. Excel DataSource Wizard



Using the second page of the wizard, you can either enter the column names and types explicitly, or use the Infer Schema button. Infer Schema will review the selected range of cells and attempt to determine the column types based on the cell types. Where the types cannot be inferred they will default to String type. You can edit the names and types after inferring, if necessary.

Note

You should not add, remove or rearrange the columns after inferring the schema, unless you make the corresponding changes to the Excel file.

Working with Excel DataSource

In this section we will look at adding an Excel DataSource with a variety of alternative ranges.

Adding Data in a Single Worksheet

Let's start with an Excel file EmpInfo.xls that consists of Emp_Id, Emp_Name, Dept_Name and Emp_Sal as shown in Figure 5.3, “EmpInfo.xls”.

Figure 5.3. EmpInfo.xls

	A	B	C	D
1	**Emp_Id**	**Emp_Name**	**Dept_Name**	**Emp_Sal**
2	1	Gerry	Electronics	7400
3	2	Michael	Electrical	8500
4	3	Frank	Computer	9000
5	4	Lawrence	Mechanical	6800
6	5	Jennifer	Sales	6500
7	6	Christopher	Electronics	8000
8	7	Flora	Computer	8600
9	8	Cathy	Electrical	7500
10	9	Peter	Sales	7000
11	10	Kendy	Mechanical	5900

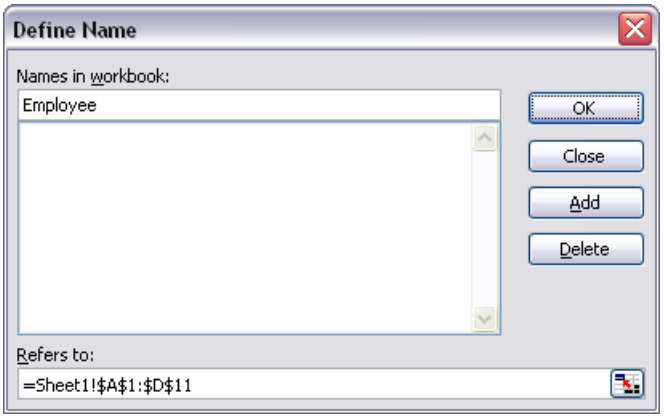
Here's how to define an Excel DataSource containing all the Employee details:

1. Before adding an Excel DataSource, the range must be defined in the Excel file.
2. Open the EmpInfo.xls file.
3. Select the Name -> Define under the Insert menu. The Define Name dialog box pops up.
4. Enter name as "Employee" and enter the range as given below in the "Refers to:" text box.

   ```
   =Sheet1!$A$1:$D$11
   ```

   Here $1 corresponds to the first cell in the file and as there are a total of ll records in this Excel file including the header. $11 corresponds to the last cell containing the final record. The Define Name dialog box appears as shown in Figure 5.4, “Define Name”. Click the OK button in the Define Name dialog box. Save the Excel file.

Figure 5.4. Define Name

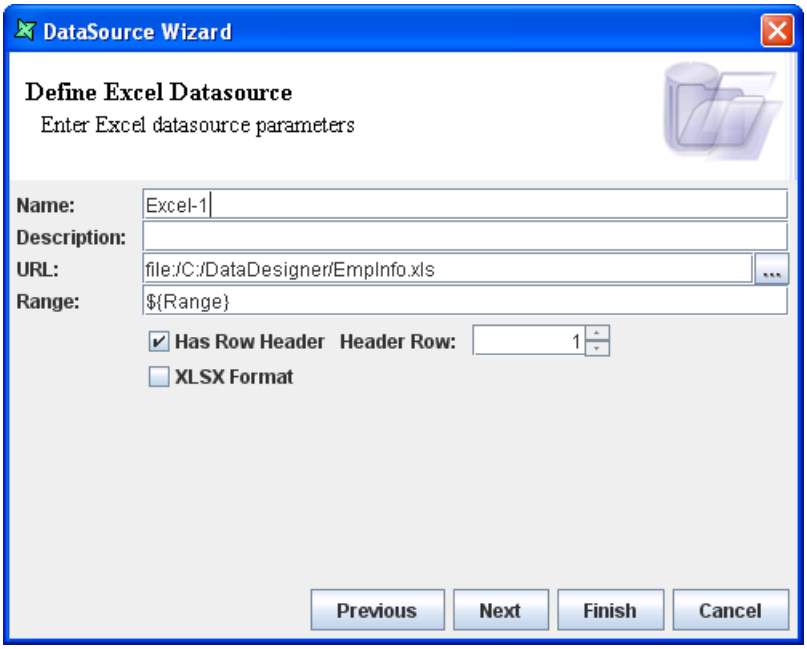


5. Launch Elixir Repertoire.
6. Choose a filesystem or folder and use the popup menu to select Add -> DataSource.
7. The DataSource Wizard appears. Select Excel DataSource and click on the Next button.
8. The Define Excel DataSource screen appears. Enter name as Excel-1.
9. Enter the URL for EmpInfo.xls in the text box provided. Alternatively, by clicking the button on the right of the text field select the EmpInfo.xls file from the File Open dialog.
10. Enter "Employee" in the Range field.

11. Select the First Row Header check box and click on the Next button.

12. The screen appears as shown in fig. Click on the Infer Schema button. Change the Data Type of Emp_Id and Emp_Sal from Double to Integer Figure 5.5, “Completed DataSource Wizard”.

Figure 5.5. Completed DataSource Wizard



13. Click the Finish button in the DataSource Wizard.

14. The Excel Data Source is added to the repository. Click the Load Data menu.

15. The Excel file is loaded displaying the employee details present in the Excel sheet as shown in Figure 5.6, “Result”.

Figure 5.6. Result

Data

[Result]

Emp_Id (Integer)	Emp_Name (String)	Dept_Name (String)	Emp_Sal (Integer)
1	Gerry	Electronics	7400
2	Michael	Electrical	8500
3	Frank	Computer	9000
4	Lawrence	Mechanical	6800
5	Jennifer	Sales	6500
6	Christopher	Electronics	8000
7	Flora	Computer	8600
8	Cathy	Electrical	7500
9	Peter	Sales	7000
10	Kendy	Mechanical	5900

Adding Data in Multiple Worksheets

This time we will work with the file EmpDetail.xls. This consists of employee details in three worksheets as shown in Figure 5.7, “EmpDetail.xls”.

Figure 5.7. EmpDetail.xls

	A	B	C	D
1	Emp_Id	Emp_Name	Dept_Name	Emp_Sal
2	1	Gerry	Electronics	7400
3	2	Michael	Electrical	8500
4	3	Frank	Computer	9000
5	4	Lawrence	Mechanical	6800
6	5	Jennifer	Sales	6500
7	6	Christopher	Electronics	8000
8	7	Flora	Computer	8600
9	8	Cathy	Electrical	7500
10	9	Peter	Sales	7000
11	10	Kendy	Mechanical	5900

Sheet1 Sheet2 Sheet3

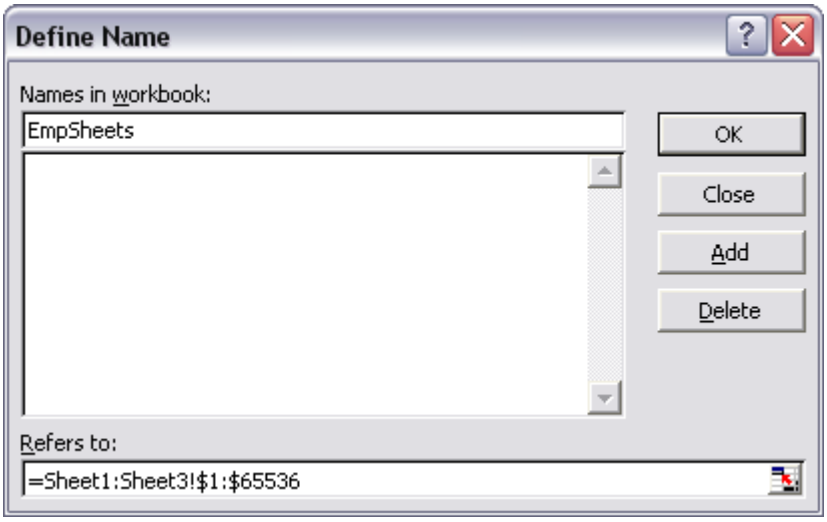
Here's how to get the values across all the sheets:

1. Before adding an Excel DataSource, the range must be defined in the Excel file.
2. Open Empdetail.xls file.
3. Select the Name -> Define under the Insert menu. The Define Name dialog box pops up.
4. Enter name as "EmpSheets" and enter range as given below in the "Refers to:" text box

   ```
   =Sheet1:Sheet3!$A$1:$D$11
   ```

   The above syntax includes the cells in sheet1, sheet3 and all the sheets between them. After setting the values the Define Name dialog window appears as shown in Figure 5.8, “Define Name”. Click the OK button in the Define Name dialog box and save the Excel file.

Figure 5.8. Define Name



5. Launch Elixir Data Designer.
6. Choose a filesystem or folder and use the popup menu to select Add -> DataSource.
7. The DataSource Wizard appears. Select Excel DataSource and click on the Next button.
8. The Define Excel DataSource screen appears. Enter name as Excel-2.

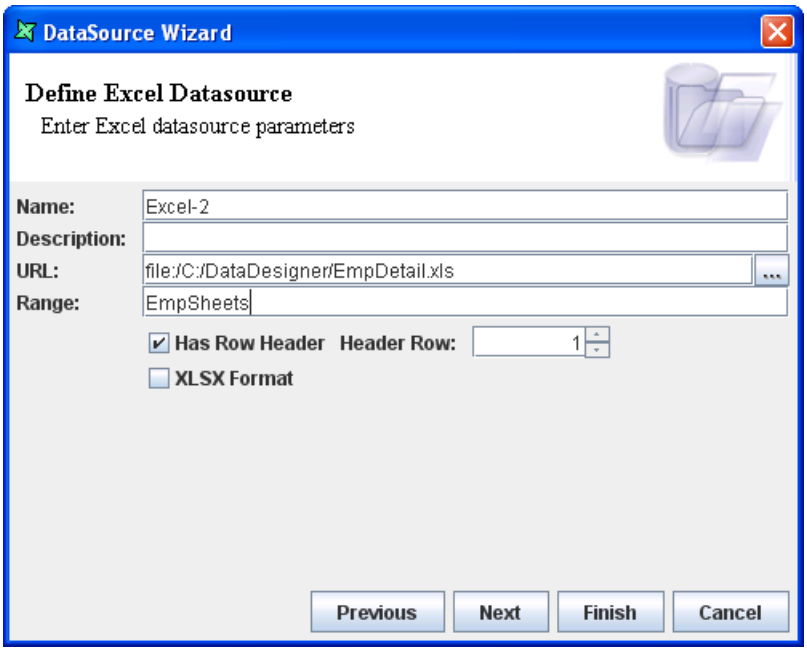
9. Enter the URL for EmpDetail.xls in the text box provided. Alternatively, by clicking the button on the right of the text field select the EmpDetail.xls file from the File Open dialog.

10. Enter "EmpSheets" in the Range field.

11. Select the First Row Header check box and click on the Next button.

12. Click on the Infer Schema button in the screen shown in fig. Change the Data Type of Emp_Id and Emp_Sal from Double to Integer. Figure 5.9, “Completed DataSource Wizard”.

Figure 5.9. Completed DataSource Wizard

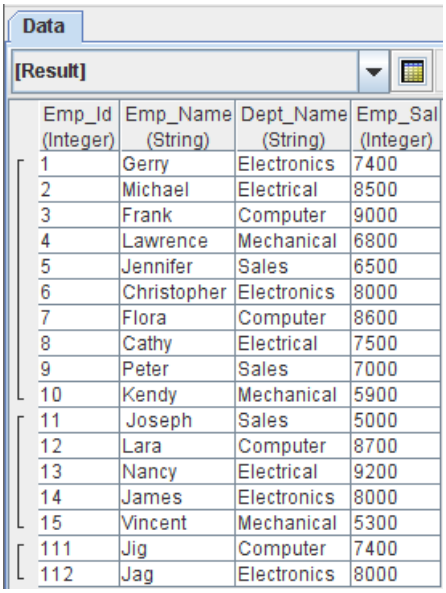


13. Click the Finish button in the DataSource Wizard.

14. The Excel DataSource is added to the repository. Double click on the Excel-2.ds data source. Click the Load Data menu.

15. The Excel file is loaded displaying the employee details present in all the Excel sheets as shown in Figure 5.10, “Result”.

Figure 5.10. Result


Data

[Result]

Emp_Id (Integer)	Emp_Name (String)	Dept_Name (String)	Emp_Sal (Integer)
1	Gerry	Electronics	7400
2	Michael	Electrical	8500
3	Frank	Computer	9000
4	Lawrence	Mechanical	6800
5	Jennifer	Sales	6500
6	Christopher	Electronics	8000
7	Flora	Computer	8600
8	Cathy	Electrical	7500
9	Peter	Sales	7000
10	Kendy	Mechanical	5900
11	Joseph	Sales	5000
12	Lara	Computer	8700
13	Nancy	Electrical	9200
14	James	Electronics	8000
15	Vincent	Mechanical	5300
111	Jig	Computer	7400
112	Jag	Electronics	8000

Passing Dynamic Parameters

To control the behavior of Elixir Data Designer dynamically, dynamic parameters are used. The parameters are specified in place of a value or values while setting the properties of a datasource or Composite element, and the value is assigned during the generation process. The generation process may prompt for parameter entry.

Parameters are defined as name-value pairs, where the name must be unique in the scope of the parameter. Refer to Appendix A, *Dynamic Parameters* for further information.

If we need a different range of data to be loaded each time, then changing the ranges values in the DataSource Wizard will be a time consuming process. Dynamic parameters can be specified for the Range in the DataSource Wizard and the values are obtained when the DataSource is used.

The file EmpDet.xls consists of employee details in two worksheets as shown in Figure 5.11, “EmpDet.xls”.

If you want to fetch the values in all the sheets of the Excel file using different range values, here's the process:

Figure 5.11. EmpDet.xls

	A	B	C	D
1	**Emp_Id**	**Emp_Name**	**Dept_Name**	**Emp_Sal**
2	1	Gerry	Electronics	7400
3	2	Michael	Electrical	8500
4	3	Frank	Computer	9000
5	4	Lawrence	Mechanical	6800
6	5	Jennifer	Sales	6500
7	6	Christopher	Electronics	8000
8	7	Flora	Computer	8600
9	8	Cathy	Electrical	7500
10	9	Peter	Sales	7000
11	10	Kendy	Mechanical	5900

Sheet1 Sheet2

1. Before adding an Excel DataSource, the ranges must be defined in the Excel file.

2. Open the EmpDet.xls file.

3. Select Name -> Define under the Insert menu. The Define Name dialog box pops up.

4. Enter name as Emp1_Small and enter range as given below in the "Refers to:" text box and click the Add button.

```
=Sheet1!$A$1:$D$6
```

5. Enter name as Emp1_All and the range as given below in the "Refers to:" text box and click the Add button.

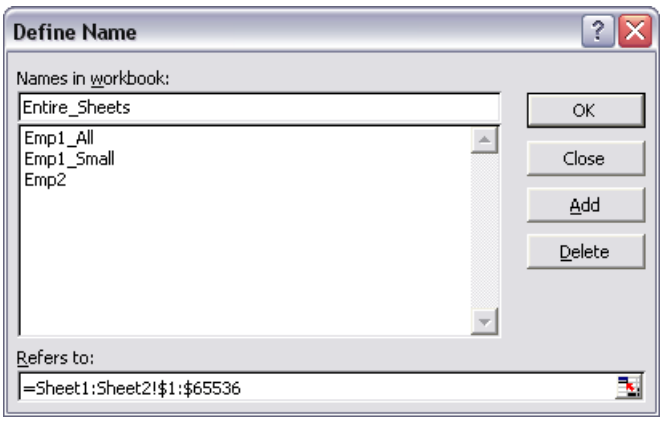
```
=Sheet1!A$1:D$11
```

6. Enter name as Emp2 and the range as given below in the "Refers to:" text box and click the Add button.

```
=Sheet2!$A$1:$D$3
```

7. Enter name as Entire_Sheets and the range as given below in the "Refers to:" text box and click the Add button. After adding the columns the dialog window appears as shown in Figure 5.12, “Define Name”. Click the Ok button in the Define Name dialog window.

Figure 5.12. Define Name




```
=Sheet1:Sheet2!A$1:D$11
```

8. Save the EmpDet.xls file.

9. Launch Elixir Repertoire.

10. Using the filesystem or folder popup menu, select Add -> DataSource.

11. The DataSource Wizard appears. Select Excel DataSource and click on the Next button.

12. The Define Excel DataSource screen appears. Enter name as Excel-3.

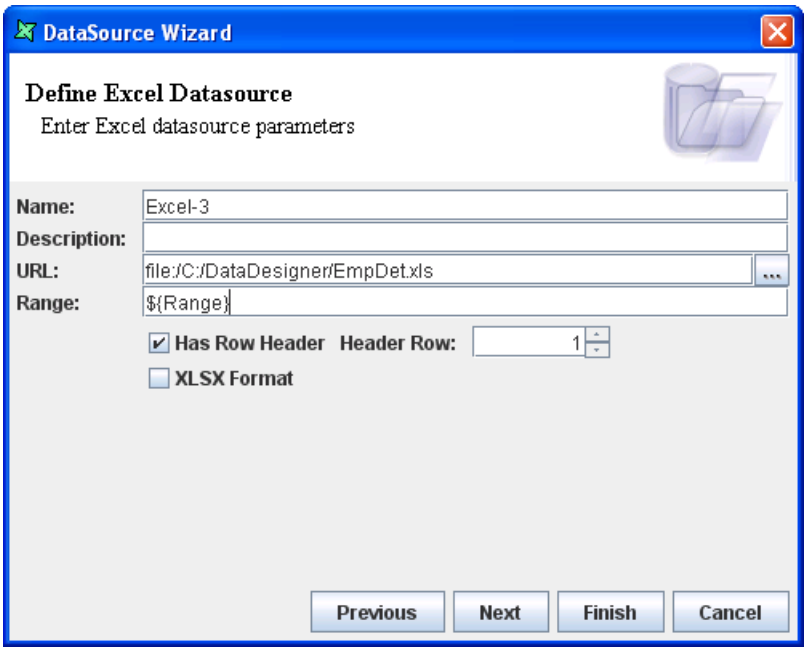
13. Enter the EmpDet.xls URL in field provided. Alternatively, by clicking the button on the right of the text field, select the EmpDet.xls file from the Open dialog.

14. Enter ${Range} in the Range field.

15. Select the First Row Header check box and click on the Next button.

16. In the screen that appears click on the Infer Schema button. Enter any range specified above in the Dynamic Parameters dialog box that appears and click the Finish button. The schema will be inferred. Change the Data Type of Emp_Id and Emp_Sal to Integer. Finally, click on the Finish button.Figure 5.13, “Completed DataSource Wizard”.

Figure 5.13. Completed DataSource Wizard

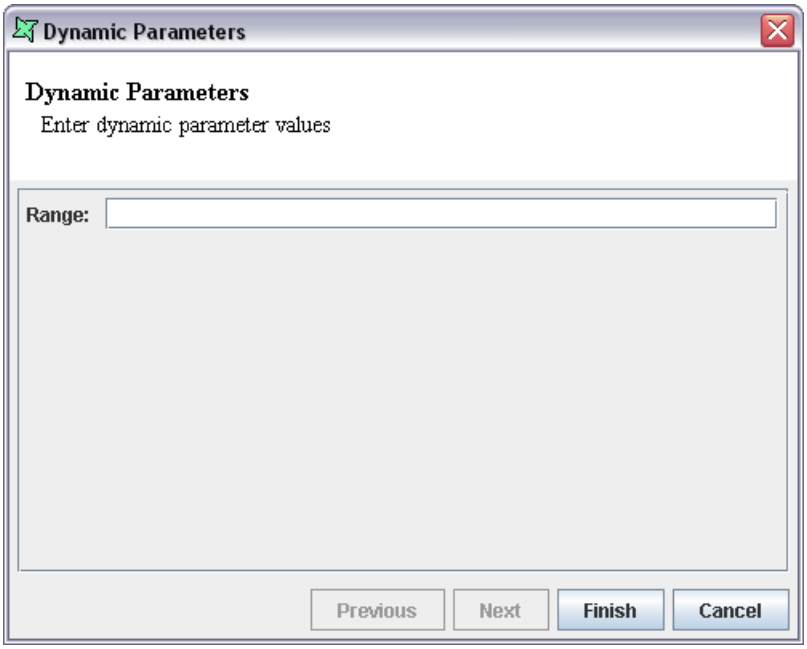


17. Click the Finish button in the DataSource Wizard.

18. The Excel data source is added to the repository. Double click on Excel-3.ds. Click the Load Data menu.

19. The Dynamic parameter dialog box pops up as shown in Figure 5.14, “Dynamic Parameters”. Enter one of the range names previously defined (Emp1_Small, Emp1_All etc.). Click the Finish button.

Figure 5.14. Dynamic Parameters



20. The Emp1_Small range is used to fetch only selected part of the data from sheet1 of the Excel file as shown in Figure 5.15, “Emp1_Small Result”. The Emp1_All range is used to fetch all the data in Excel file sheet1 as shown in Figure 5.16, “Emp1_All Result”. Similarly, the Emp2 range is used to fetch only selected part of the data from sheet2 of the Excel file and the Entire_Sheets range are used to fetch all the records from both the sheets.

Figure 5.15. Emp1_Small Result

Data

[Result]

Emp_Id (Integer)	Emp_Name (String)	Dept_Name (String)	Emp_Sal (Integer)
1	Gerry	Electronics	7400
2	Michael	Electrical	8500
3	Frank	Computer	9000
4	Lawrence	Mechanical	6800
5	Jennifer	Sales	6500

Figure 5.16. Emp1_All Result

Data

[Result]

Emp_Id (Integer)	Emp_Name (String)	Dept_Name (String)	Emp_Sal (Integer)
1	Gerry	Electronics	7400
2	Michael	Electrical	8500
3	Frank	Computer	9000
4	Lawrence	Mechanical	6800
5	Jennifer	Sales	6500
6	Christopher	Electronics	8000
7	Flora	Computer	8600
8	Cathy	Electrical	7500
9	Peter	Sales	7000
10	Kandy	Mechanical	5900

Chapter 6
Text DataSource

Overview

In Elixir Data Designer the data from a text file can be accessed using the Text DataSource.

The tool provides encoding support, which includes multi-lingual formats, so text data sources which are encoded in different formats can be added. Also, text data sources with fields having different date, time and timestamp formats can be added.

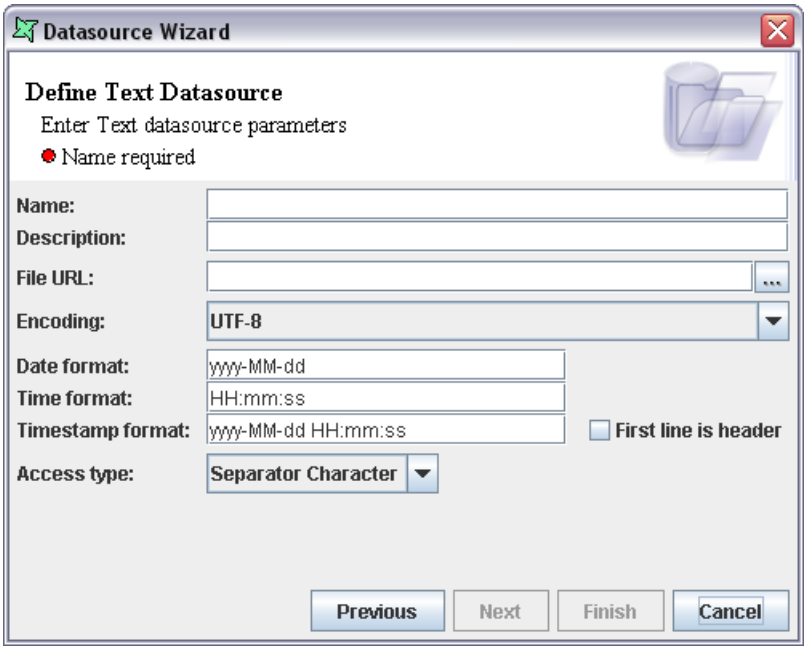
A Text DataSource can be created simply by right-clicking on an CSV file on the repository and select “Define as a Text DataSource”.

In addition to straightforward field-based access, regular expressions are supported to give power users very flexible filtering and extraction options.

Text DataSource Wizard

The first Text DataSource properties screen is shown in Figure 6.1, “Define Text DataSource”.

Figure 6.1. Define Text DataSource



The Name of the data source is entered in the Name text box and any extra description that is used to describe the data source can be entered in the Description text box.

By default, the text files are in ASCII format. If a text file with an alternate encoding needs to be accessed then the encoding type should be selected from the Encoding combo box.

The format of any dates in the text file is entered in the Date format text field.

The format of any times in the text file is entered in the Time format text field.

The format of any timestamps (date and time) in the text file is entered in the Timestamp format text field.

By default the date, time and timestamp formats are specified according to ISO standards.

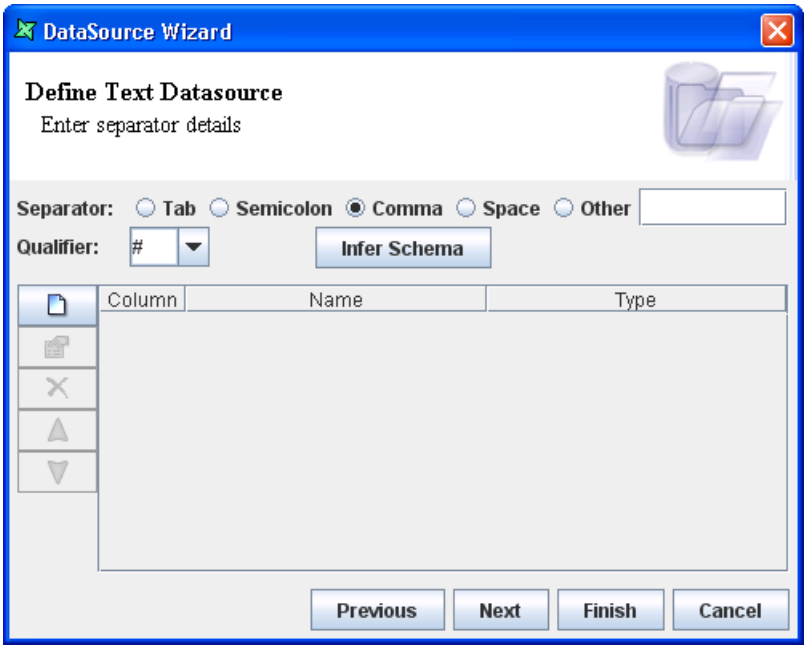
The URL of the file containing the text is specified in the File URL text box. Alternatively, by clicking the button on the right of the text box, the file can be selected from the Open dialog window. Note that this URL can access not only local files, but any files accessible through HTTP, FTP or from the repository (use the repository: prefix).

There are three different ways of extracting data from a text file: Separator character, Fixed Width and Regular Expression.

Separator Character

On selecting the Separator Character access type and clicking the Next button the screen as shown in Figure 6.2, “Separator Type Properties” appears. The separator details are entered in this screen.

Figure 6.2. Separator Type Properties



The type of separator is chosen from among the different Separator options. If there is any special type of separator other than those specified then that particular separator must be entered in the "Others" text box. The type of qualifier used in the text file can be selected or entered in the combo box.

The Infer Schema button is used to infer the schema.

On clicking the Add Column button the Add Column dialog window pops up.

The name of the new column is entered in the text box and the data type of the field is selected from the combo box. The new column is added to the data source.

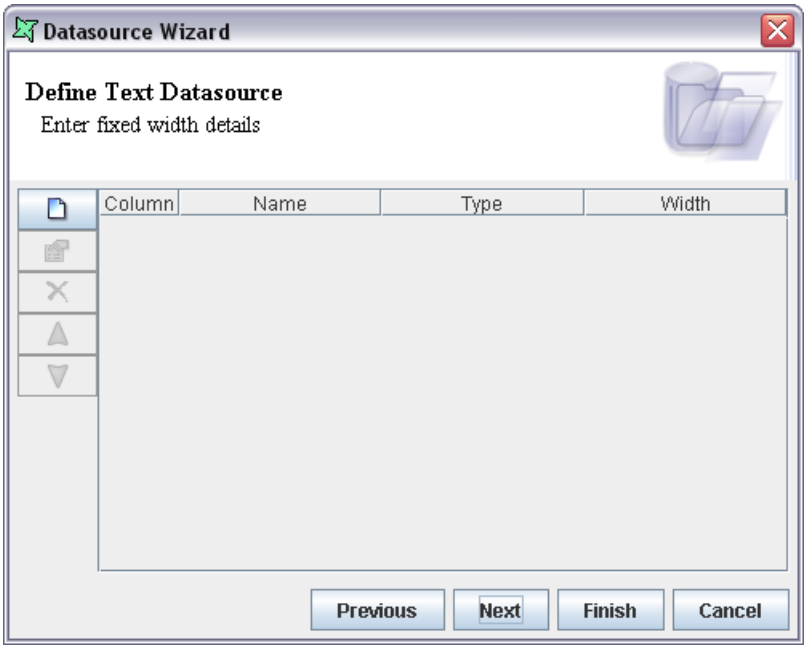
The column can be edited by selecting the Edit Column button. The Move Up, Move Down are used to reorder the columns. The Remove Column button is used to delete a column.

Fixed Width

On selecting the Fixed Width access type and clicking the Next button the screen appears as shown in Figure 6.3, “Fixed Width Type Properties”.

On clicking the Add Column button the Add Column dialog window pops up.

Figure 6.3. Fixed Width Type Properties



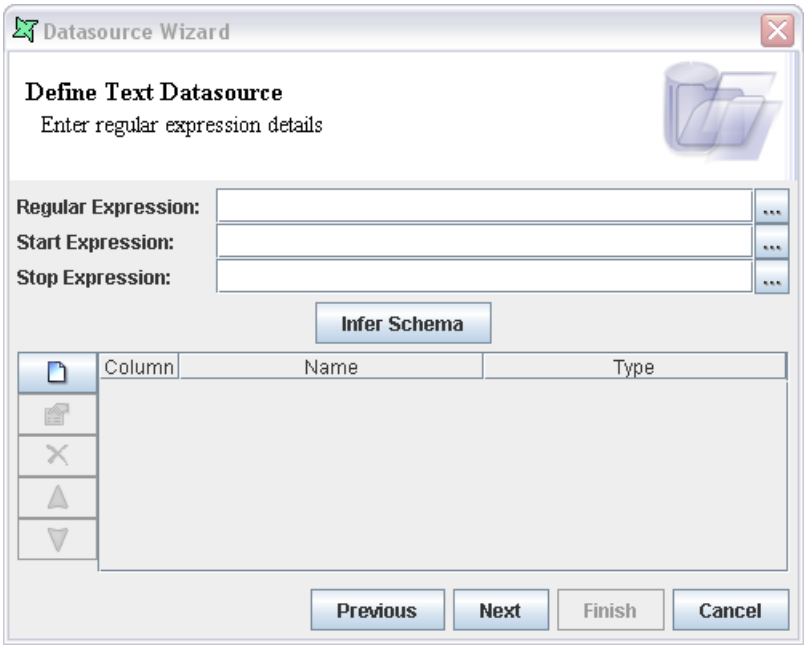
The name of the column is entered in the text box and the data type of the field is selected from the combo box. The width of the column is entered in the text box. On clicking the OK button the new column is added to the data source.

Using the Edit Column, Move Up, Move Down and Remove Column button the columns can be edited, re-ordered or deleted as necessary.

Regular Expression

On selecting the Regular Expression access type and clicking the Next button the screen appears as shown in Figure 6.4, “Regular Expression Type Properties”.

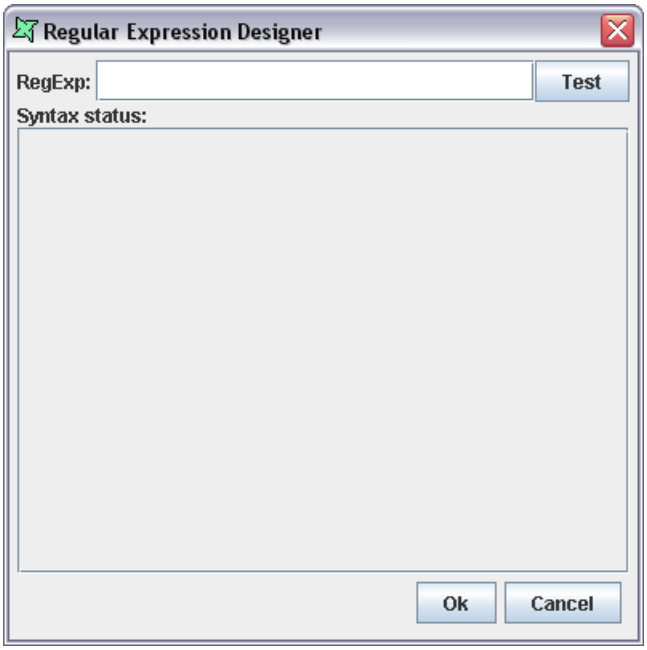
Figure 6.4. Regular Expression Type Properties



In this window the Regular Expression can be entered in the Regular expression text box.

Alternatively, by clicking the button to the right of the text box a separate designer window appears as shown in Figure 6.5, “Regular Expression Designer”.

Figure 6.5. Regular Expression Designer



The Regular expression is entered in the RegExp text box and the Test button is clicked to test the validity of the expression. If the expression is not valid then syntax errors are displayed. If there are no syntax errors then the fields are displayed in the window. Finally, on clicking the Ok button, the expression is assigned in the Regular Expression text box.

Regular expression processing requires the entry of up to three regular expressions. The first one, named Regular Expression is mandatory. This selects which records are included in the result and how

the field values are extracted from those records. For example, a regular expression such as "^abc(.*)" will only pass through those records where the line begins with abc (^ marks the beginning of the line) and extract a single field, matched by the rest of the line (.*) meaning a group which contains zero or more characters). Similarly, "(.*),(.*)" will extract two fields, separated by a comma. In this case, all lines will be used, as there is no filter criteria, like the "^abc" in the earlier example. There are plenty of books and on-line resources which will give full details of regular expression syntax.

So far we have only looked at one regular expression. There are two more: Start and Stop. These are optional, but if supplied will turn on, and off record processing. If a Start expression is supplied, all lines will be discarded until this Start expression is matched. Subsequent lines will be processed as described previously until the Stop expression (if supplied) is matched. Subsequent lines will again be discarded. A brief example will show how useful this is.

```
Summary Jon,Total Bill,Total Details Jon Jon,First
  Jon,Second Bill Bill,First Bill,Second Bill,Third Comments
  None,Nothing
```

Given the above text file, we can extract just the details by using these three expressions: Start = "Details", Stop = "Comments", Regular Expression = "(.*),(.*)". Of course, you leave out the quotation marks when entering expressions into the text fields. This configuration will ignore all the Summary values (even though they match the Regular Expression criteria of two fields separated by a comma). Once we reach the line which matches the start criteria "Details" the Regular Expression matching starts. The next line just contains "Jon", which doesn't match the Regular Expression requirements and so is discarded. "Jon,First" and "Jon,Second" do match, so they are passed through as records. "Bill" doesn't match, so it is skipped, but "Bill,First", "Bill,Second" and "Bill,Third" are passed through. Finally, we reach a line which matches the Stop expression, so subsequent line processing is turned off (which means "None,Nothing" is discarded). To summarize, five records, each containing two fields are extracted:

```
Jon,First Jon,Second Bill,First Bill,Second
  Bill,Third
```

The Start Expression is entered in the Start Expression text box. Alternatively, on clicking the button to the right of the text box the Regular Expression Designer Window appears. The Start Expression can be entered in the RegExp text box, tested and added.

The Stop Expression is entered in the Stop Expression text box. Alternatively, on clicking the button to the right of the text box the Regular Expression Designer Window appears. The Stop Expression can be entered in the RegExp text box, tested and added.

The Infer Schema button is used to infer the text data source columns based on the number of groups in the regular expression.

Working with Text DataSource

In this section we will discuss the different ways of adding the text data sources.

Using Separator Characters

If you are given a text file in which the fields are enclosed within quotes and are separated by semicolon you can easily add it as a datasource. In this example we will use the EmpInfo.txt file from the Elixir Data Designer samples.

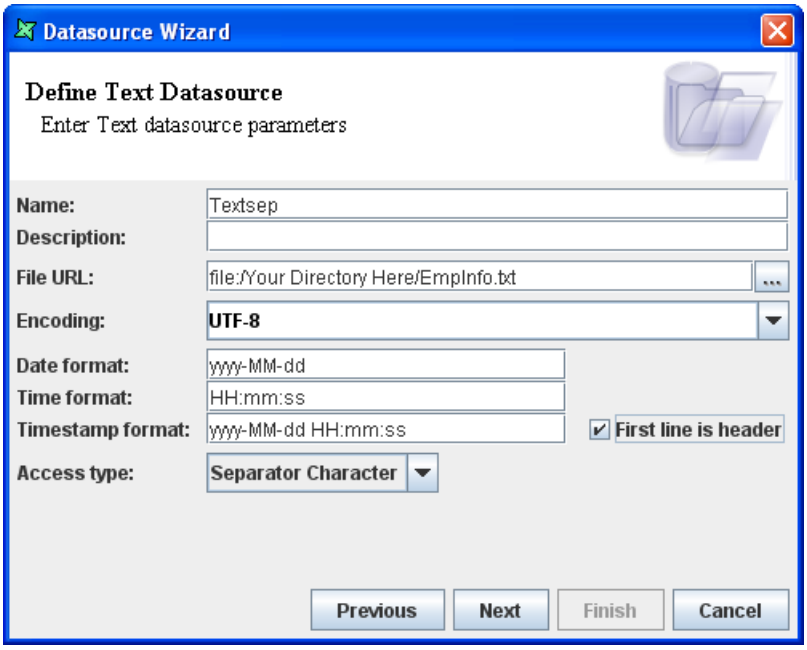
1. Choose a file system or folder and use the popup menu to select Add -> DataSource.

2. The DataSource Wizard appears. Select the Text DataSource and click the Next button.

3. The "Define Text DataSource" screen appears.

4. Enter a unique name, such as `Textsep`.
5. Enter the URL in the text box provided. Alternatively, by clicking the button to the right of the text field, select the file from the Open dialog window.
6. Select ASCII as the Encoding option as the EmpInfo.txt file is in ASCII format.
7. The Date format, Time format and Timestamp format text fields can have default values.
8. Select the First line is header check box.
9. Select the Separator Character Access type. After setting the properties the screen appears as shown in Figure 6.6, “Sample Text DataSource”. Click the Next button.

Figure 6.6. Sample Text DataSource



10. Select Semicolon as the Separator.
11. Select " option from the Qualifier combo box.
12. Click the Infer Schema button and click the Finish button.
13. You can then open the Textsep.ds data source and click Load Data to view the output.

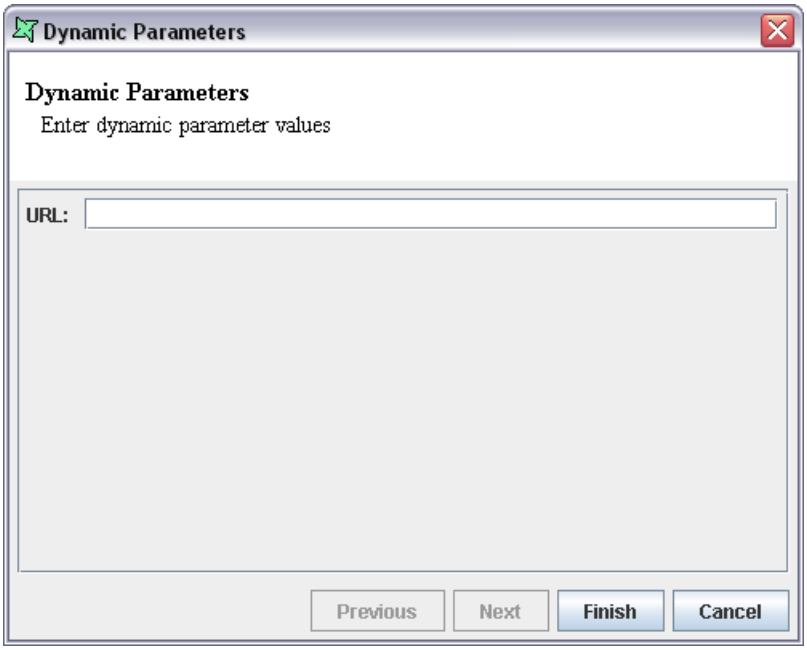
Defining a URL with a Dynamic Parameter

Here's how to add a text file by passing a dynamic parameter to the URL.

1. Open the Add -> DataSource wizard to create a Text DataSource as before..
2. On the "Define Text DataSource" screen, enter a unique name such as `Text_URL`.
3. Enter the URL as ${URL}. This indicates a dynamic parameter will be inserted here when we use the data source.
4. Follow the rest of the steps as described in the previous section.

5. You will find that on clicking the Infer Schema button a dialog appears as shown in Figure 6.7, “Dynamic Parameters”.

Figure 6.7. Dynamic Parameters



6. Enter the URL as given below (modify according to the location of your EmpInfo.txt file - remember in URLs all slashes are '/' even on Microsoft operating systems).

```
file:/C:/EmpInfo.txt
```

7. The columns in the data source are listed in the window. Click the Finish button.

8. The Text_URL data source is added to the repository. Select and double click on the Text_URL.ds data source. On clicking the Load Data menu the Dynamic Parameters dialog window appears. Enter the URL in the text box. On clicking the Finish button the output is displayed. You can now substitute any other file URL that conforms to the same schema (ie. has the same number and type of fields) and the data source will load it correctly.

Using Fixed Width Columns

Here's how to add a text file having columns of fixed width.

1. Use the file system or folder popup menu to select Add -> DataSource.

2. The DataSource Wizard appears. Select the Text DataSource and click the Next button.

3. The "Define Text DataSource" screen appears.

4. Enter a unique name, such as `Text_Width`.

5. Enter the URL in the text box provided. Alternatively, click click the button to the right of the text field to invoke the chooser. In this case we will select the DateJoin.txt file.

6. Select the First Line is Header check box.

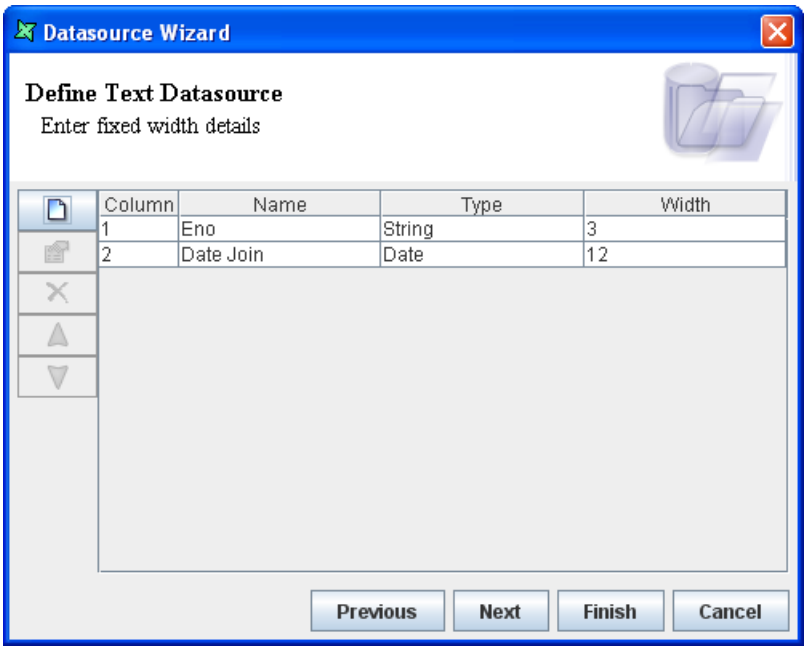
7. Select Fixed Width as the Access type and click the Next button.

8. Add a column named `Eno` with data type String and width 3.

9. Add a second column named `Date Join` with type String and width 12.

10. The columns are added to the Wizard. The screen appears as shown in Figure 6.8, “Fixed Width Sample”. Click the Finish button.

Figure 6.8. Fixed Width Sample

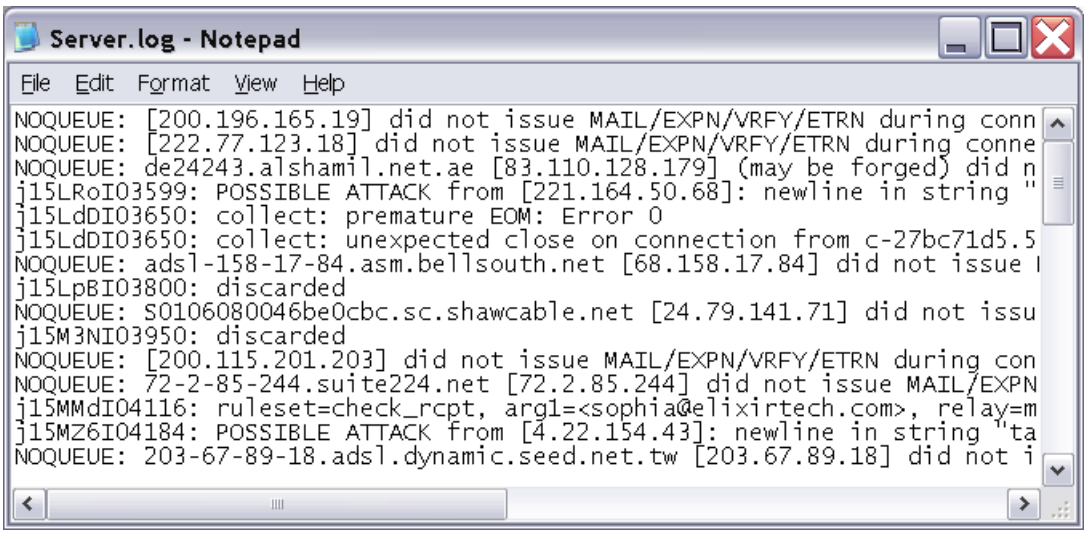


11. The Text DataSource is added to the repository. Select and double click on the Text_Width.ds data source. By clicking the Load Data menu you can verify the output.

Using Regular Expressions

Here's how to extract data from a log file like Figure 6.9, “Log file”. This can be achieved easily by using the Regular Expression Access type:

Figure 6.9. Log file

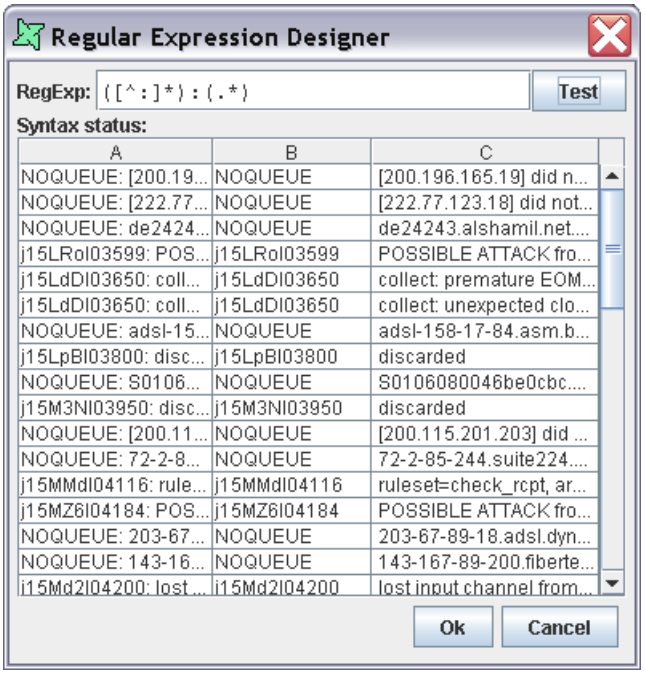

Server.log - Notepad
File Edit Format View Help
NOQUEUE: [200.196.165.19] did not issue MAIL/EXPN/VRFY/ETRN during conn
NOQUEUE: [222.77.123.18] did not issue MAIL/EXPN/VRFY/ETRN during conne
NOQUEUE: de24243.alshamil.net.ae [83.110.128.179] (may be forged) did n
j15LRoI03599: POSSIBLE ATTACK from [221.164.50.68]: newline in string "
j15LdDI03650: collect: premature EOM: Error 0
j15LdDI03650: collect: unexpected close on connection from c-27bc71d5.5
NOQUEUE: adsl-158-17-84.asm.bellsouth.net [68.158.17.84] did not issue
j15LpBI03800: discarded
NOQUEUE: S0106080046be0cbc.sc.shawcable.net [24.79.141.71] did not issu
j15M3NI03950: discarded
NOQUEUE: [200.115.201.203] did not issue MAIL/EXPN/VRFY/ETRN during con
NOQUEUE: 72-2-85-244.suite224.net [72.2.85.244] did not issue MAIL/EXPN
j15MMdI04116: ruleset=check_rcpt, arg1=<sophia@elixirtech.com>, relay=m
j15MZ6I04184: POSSIBLE ATTACK from [4.22.154.43]: newline in string "ta
NOQUEUE: 203-67-89-18.adsl.dynamic.seed.net.tw [203.67.89.18] did not i

1. Use the filesystem or folder popup menu to select Add -> DataSource.
2. The DataSource Wizard appears. Select the Text DataSource and click the Next button.
3. The "Define Text DataSource" screen appears.
4. Enter a unique name such as `Text_RegExp`.
5. The URL in this case will be the path to Server.log (you can use the Open dialog to find it).
6. Select Regular Expression as the Access type and click the Next button.
7. The "Define Text DataSource" screen appears. The Regular Expression details are entered in this screen.
8. Click the "..." button on the right of the Regular Expression text box. The Regular Expression Designer Window appears.
9. In the RegExp text box enter the Regular Expression as given below

   ```
   ([^:]*):(.*)
   ```

   This breaks the log into two chunks. The first part "([^:]*)" reads up to the first colon. The second part "(.*)" reads everything else. There is an explicit colon in the middle, so this is not part of either chunk. See a regular expression reference for more help in interpreting this syntax.
10. Click on the Test button to check the syntax status. If there are no errors in the syntax then the columns are displayed in the window of the designer as shown in Figure 6.10, “Regular Expression Designer”. The first column shows the original line, the subsequent two columns (in this case) shown the portion of text that matches the groups in the expression. On clicking the Ok button the syntax is assigned in the Regular Expression text box.

Figure 6.10. Regular Expression Designer



11. On clicking the Infer Schema button the columns are inferred based on the groups in the regular expression.

12. Click the Finish button. The Text_RegExp data source is added to the repository.

13. Select and double click on the Text_RegExp.ds data source. Use the Load Data button to ensure the output is correct. We have split the data into two fields, but this particular log has two different formats embedded in it. The text at the bottom, below "**Unmatched Entries**" is in a different format. Notice that the line "**Unmatched Entries**" is not in the output, because it doesn't include a colon, so it is automatically discarded. We can discard the lines below this with a Stop expression.

Using Start and Stop Expressions

Here's how to add a log file using only part of the data from the file. To do this the Start Expression and Stop Expressions are used along with the Regular Expression.

1. Create a data source as described in the previous example, or modify the one you've already created.

2. In the Regular Expression designer window click the "..." button to the right of the Stop Expression text box. The Regular Expression designer window appears.

3. In the Stop RegExp text box enter "**Unmatched Entries**" (without the quotes). You will notice a syntax error is reported because * is a special character in regular expressions. Insert a backslash before each * to have them handled literally: "**Unmatched Entries**". This will remove the syntax error warning and you can click on the Test button. Only one record will be found by this expression, the line which marks the start of the second log section. On clicking the Ok button the syntax is assigned in the Start Expression text box.

4. On clicking the Infer Schema button the columns are inferred as before.

5. Click the Finish button.

If you view the output of this data source, you will see that only the lines up to the Stop expression are processed, those with a different format below are now skipped. If you wanted only those items below, you could move the Stop expression to the Start expression, and now only records after the Start matches would be processed. The Text DataSource can support the extraction of multiple chunks by using combinations of start and stop.

Chapter 7
XML DataSource

Overview

In Elixir Data Designer data in XML files can be accessed by adding an XML DataSource.

The basic requirements to access the XML document is a well formed XML document and a XPath that is used to query the XML document, locate and retrieve data.

The text data sources with fields having different date, time and timestamp formats can be added.

Furthermore, the URL of the files can be specified in the form of an XQuery to retrieve data from XML databases like Software AG Tamino or EAI solutions with XML output like BEA Liquid Data.

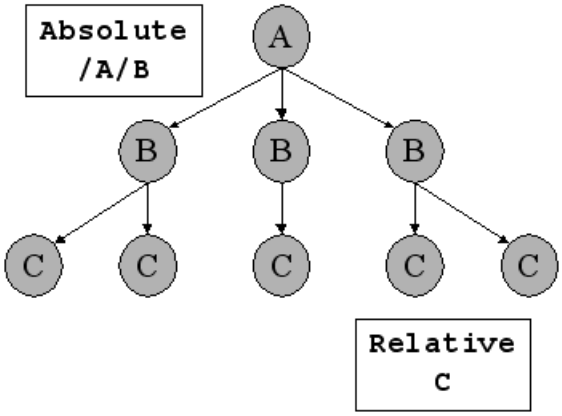
XPath

XPath is a W3C specification for retrieving data from an XML file. It is a query language for XML. Elixir Data Designer provides an XPath Builder. The features are given below:

- It provides a tree view structure of the XML source.
- Specifies the target "records" with Root XPath.
- Specifies columns to be added by dragging and dropping from the XML tree.
- The column name or the XPath of the columns can be changed.
- It provides full support for XPath.

The XML tree structure appears as shown in Figure 7.1, “An XML Tree”.

Figure 7.1. An XML Tree



Here A is the main element having sub-element B which in turn has sub-element C. There are two root paths as given below

- *Absolute Path*: The absolute path starts from the root of the document.

- *Relative Path*: The relative path starts from each of the currently defined node(s).

Some basic syntax of the XPath are given below:

- /A/B/C - All the C children of B which are children of the root node A.
- /C - The C root node.
- //C- All C nodes in the document. The Name of the file is entered in the Name column. The Description of the file is specified in the Description text box.
- id("S_24") - The node with ID value "S-24".

An absolute XPath starts from the root of the document tree. As shown above an XPath can be absolute or relative.

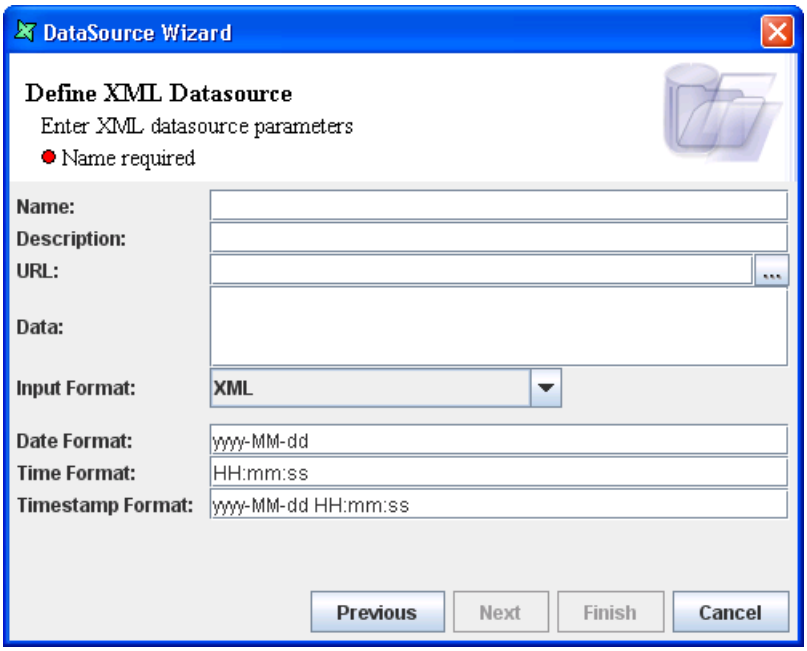
Note

For more information on XPath refer to the website http://www.w3schools.com/xpath/xpath_functions.asp

XML DataSource Wizard

The XML DataSource Wizard properties are shown in Figure 7.2, “Define XML DataSource”.

Figure 7.2. Define XML DataSource



The unique name of the data source is entered in the Name text box and any extra description that is used to describe the data source can be entered in the Description text box.

The format of date, time and timestamp strings in the XML can be specified. By default the date, time and timestamp formats follow ISO standards. If the date and time fields in the XML file follow a different format then that format has to be defined in the appropriate text boxes.

XML data can be supplied in one of two ways. You can either enter the URL of an XML file, or enter the XML directly into the wizard. The URL approach should be preferred when the XML contents is large. Where both URL and XML are specified, the XML value will take precedence.

Note

Both the URL and XML entries allow for ${substitution} to be used. For XML this allows the XML contents to be passed in through dynamic parameters - for example from a report. To do this, set the XML text to something like ${XML} and then supply the XML parameter at render time.

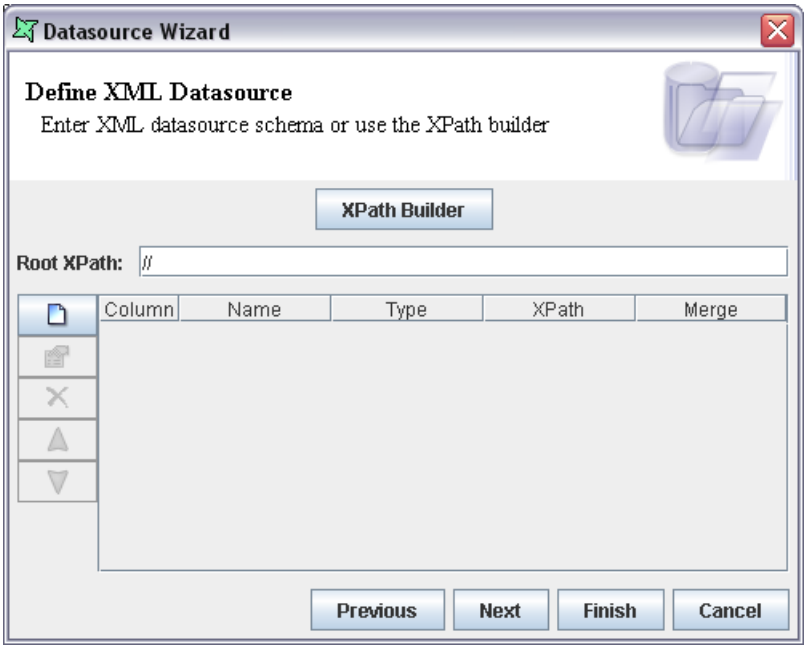
There are two ways by which the columns in the XML file can be added.

There is a main "Root XPath", which allows you to specify from which node in the tree you would want to start your traversal. The XPath is used to specify the XPath of the node, with respect to the "Root XPath".

Method 1

In this method the columns in the XML source can be added by clicking on the XPath Builder button shown in Figure 7.3, “Choosing the XPath Builder”.

Figure 7.3. Choosing the XPath Builder



There are two panel windows in the XPath Builder. The tree structure view of the XML file will be displayed in the left panel. The value of the fields selected in the XML file will be displayed in the text field below the panel.

The Root XPath is specified in the text box provided at the top of the right panel. Select the field to be displayed and then drag and drop them in the Field and XPath cells of the columns. Alternatively enter the name of the Field and XPath in the corresponding cells. Select the Merge type from the combo box. Then field values are displayed in the column.

On clicking the Ok button the selected fields are added to the Wizard.

Method 2

This method can be used if the XPath of the columns are already known. In this method the Root XPath is entered in the text box. There are five buttons. They are Add Column, Edit Column, Move Up, Move Down and Remove Column buttons.

On clicking the Add Column button, the Add Column dialog window appears. The name of the column is entered in the text box. The data type of the column is selected from the Data Type combo box.

The XPath is entered in the text box. Finally on selecting the Merge from the combo box and clicking the OK button the column is added to the Wizard.

The Edit Column button is used if the values in the column have to be altered. Using the Move Up and Move Down button the selected columns can be reordered if the sequence in which they are displayed needs to be changed. The Remove Column button is used to delete the column.

Use of Merge

Whenever a relative XPath expression returns more than one result, you have the option of how to combine the results into a single field. The usual behaviour is to concatenate, so there are merge options for using Space, Comma, Semicolon, Newline and Tab as separators between the merged string components. None will concatenate the strings directly with no separator. No-Merge will just use the first result returned and discard the rest.

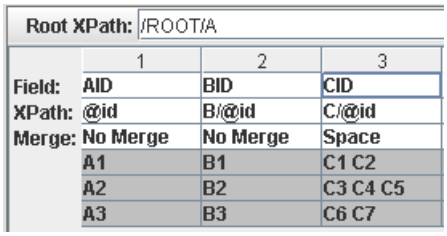
There is one final option, Group. If you choose Group then all subsequent XPaths are relative to the Group and will generate multiple records sharing all common fields. For example, given this input:

```
<?xml version="1.0"?>

<ROOT>
  <A id="A1">
    <B id="B1"/>
    <C id="C1"/>
    <C id="C2"/>
  </A>
  <A id="A2">
    <B id="B2"/>
    <C id="C3"/>
    <C id="C4"/>
    <C id="C5"/>
  </A>
  <A id="A3">
    <B id="B3"/>
    <C id="C6"/>
    <C id="C7"/>
  </A>
</ROOT>
```

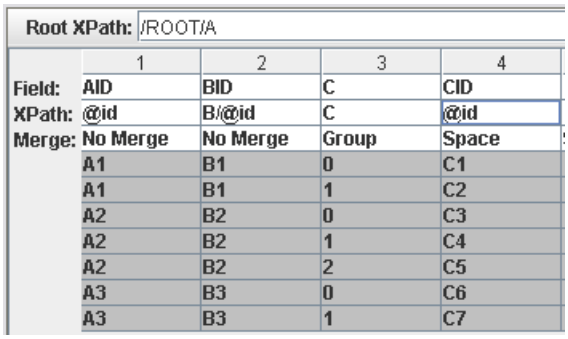
. Use of a Merge Space will concatenate all the C ids together, as shown in Figure 7.4, “Merge Space”

Figure 7.4. Merge Space



However, use of Merge Group will add extra records to ensure each C id remains distinct. This is shown in Figure 7.5, “Merge Group”.

Figure 7.5. Merge Group



Root XPath: /ROOT/A

	1	2	3	4
Field:	AID	BID	C	CID
XPath:	@id	B/@id	C	@id
Merge:	No Merge	No Merge	Group	Space
	A1	B1	0	C1
	A1	B1	1	C2
	A2	B2	0	C3
	A2	B2	1	C4
	A2	B2	2	C5
	A3	B3	0	C6
	A3	B3	1	C7

There are now two A1 records, because there are two C nodes grouped within A1. Note that the value of the Group row is a zero-based index of the number of the record within the parent.

Subtree Optimization

The XML DataSource provides an option for processing huge XML files. By default, the entire XML Document is loaded into memory, as XPath can only work fully when the tree is available. As this is not practical when the source files are large, there is a special Subtree Optimization mode that can be used instead.

When Subtree Optimization is enabled, the XML file is read and processed sequentially. This mode only works when the Root XPath expression is a simple, absolute path to the root of the subtree that represents a record. For example, "/data/customer/record" is an absolute path, this will work. However "//record", which is also a valid XPath, won't work in this mode.

Note

You must ensure the Root XPath is a simple,absolute xpath for this mode to work.

When the data source encounters an element in the XML source with the designated root path, a subtree is then constructed from that element and the descendants. Hence, full XPaths can still be used for the extraction of fields, though they only have access to the subtree. When this mode is used, only the subtree needed to process one record exists in memory at a time. This greatly reduces the memory requirements.

HTML

The XML DataSource can also be used to read HTML. In this mode, the HTML is parsed into well-formed XML that can be queried using XPath expressions. The HTML parser can handle HTTPS pages, pages that are not well-formed - eg. with missing close tags, or unquoted attributes and will clean the tree so that it can be used for effective data acquisition. HTTPS is only supported from Repertoire 7.3 onwards.

HTML parsing is not available when Subtree Optimization is enabled.

EDIFACT

EDIFACT is similar to a text file and is slightly easier to read compared to XML. From Repertoire 7.4 onwards, EDIFACT can be used to be parsed into XML. The rest of the steps to create a XML datasource will be just like the other input formats.

Search in XPath Builder

In the field with a magifying glass on the right, user can enter values, then click on the magifying glass to do a search. The name of the field and the value will be displayed in the bottom field. The field contained the value will be highlighted in the tree structure. Text and numbers are accepted in the Search field.

Working with XML DataSources

This example uses the sample file "Telephone-Mod.xml". You can either place this file in the repository, or reference it from any location on your disk. When the file contents are fairly static, it is preferable to place the file in the repository so that it can be deployed to the server and maintained along with the data source. If the file contents are dynamic, you might prefer to reference the file in situ, rather than copy it into the repository. Alternatively, you could have a web server provide the XML data on demand.

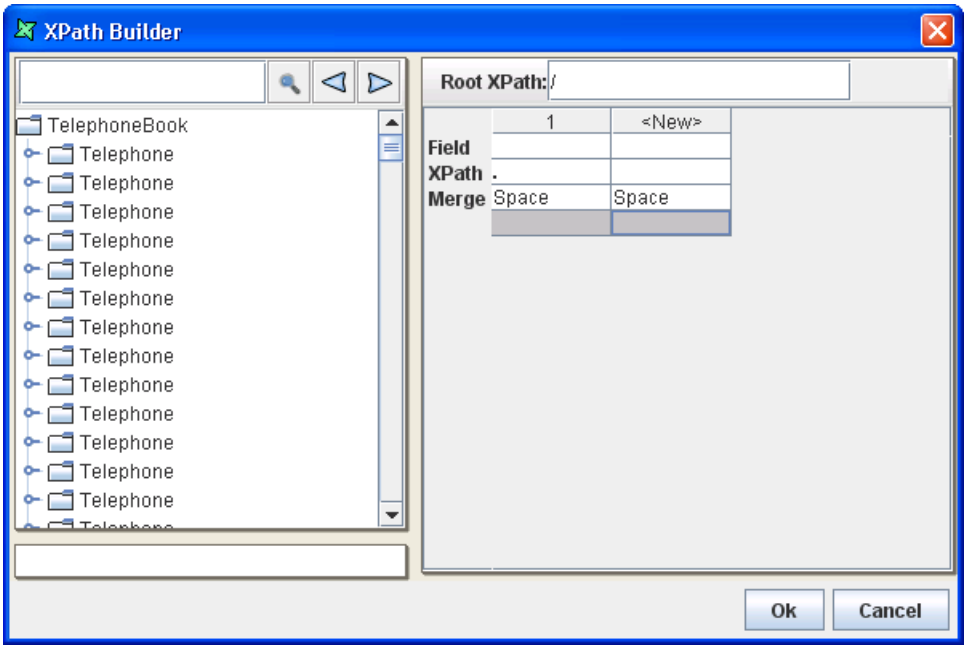
Create an XML DataSource using the Add DataSource Wizard and give it a unique name. For the URL value, you can use repository:/some-path (reads from the repository), file:/some-path (reads from any location on your machine) or http://host/some-path to get the data from a web server. You can also use the file chooser [...] to choose a file: location.

Choose Next and click the XPath Builder button. This will load the XML document from the URL you've just provided. You will see Figure 7.6, “XPath Builder”. Enter the Root XPath that will identify the record locations. For this example, use

```
/TelephoneBook/Telephone
```

which will select each of the Telephone elements that are children of Telephone book. After keying in the root XPath, press Enter.

Figure 7.6. XPath Builder



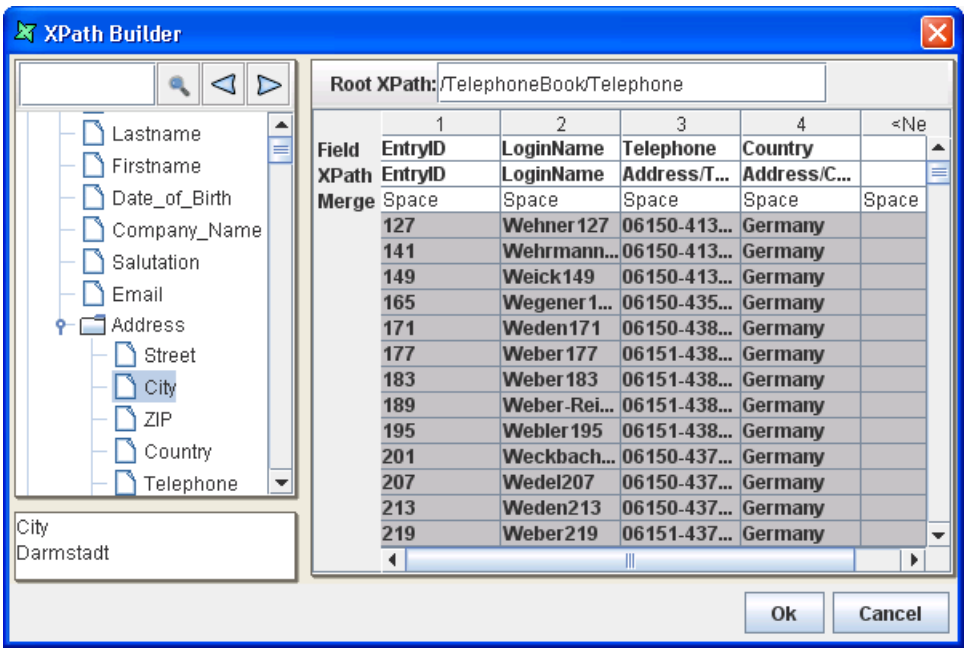
Note

In this example, there is another inner Telephone element nested within Telephone, so we can't use //Telephone or it would select those too. In any case, // XPath operations will be slower than /, so avoid them unless you need to select from a combination of depths in the XML tree.

When you press Enter, you will see a number of empty (grey) rows appear in the XPath Builder table. We now need to choose what attributes we want to extract. Expand the XML tree shown on the left to see the children of the first Telephone element. Drag the EntryID element over the first column in the table and let go. You should now see the EntryID values have been filled in to that column. Repeat the process, dragging LoginName into the <New> column. This shows the LoginName values and adds another <New> column for your next drop. You can also enter the XPath expressions directly into the table if you need more complex constructs.

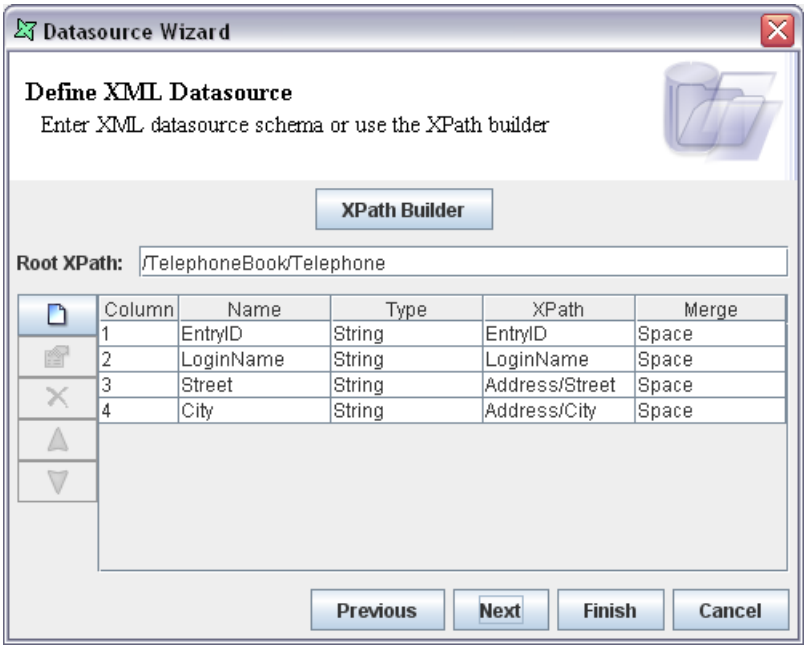
The next elements we want are the Street and City from inside Address. Therefore, expand the Address element and drag these in turn over the <New> column. You should now see Figure 7.7, “XPaths Completed”. Note the builder has inserted the correct XPaths: Address/Street and Address/City for you, based on the Root XPath we defined earlier.

Figure 7.7. XPaths Completed



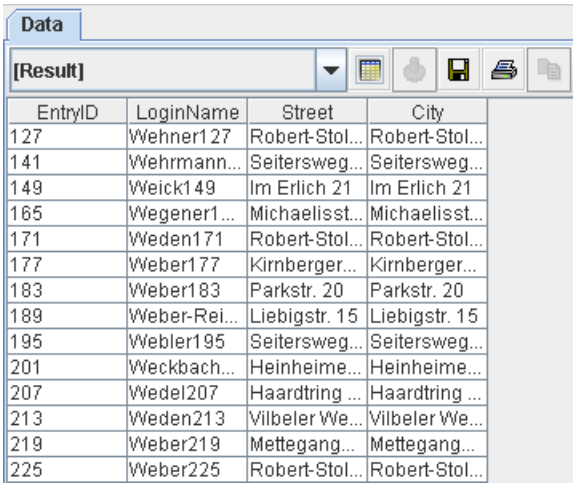
You can now choose Ok to close the XPath Builder. Back in the wizard you can see in Figure 7.8, “XML DataSource Schema”, the chosen fields along with the default type String, which is correct in this case. If your values are integers or dates etc. you can adjust the data types (or any of the other information) here.

Figure 7.8. XML DataSource Schema



When the wizard has closed, the new data source will be loaded into the workspace. You can test it by clicking the Load Data button. The output should appear as shown in Figure 7.9, “DataSource Results”.

Figure 7.9. DataSource Results



Chapter 8
Properties DataSource

Overview

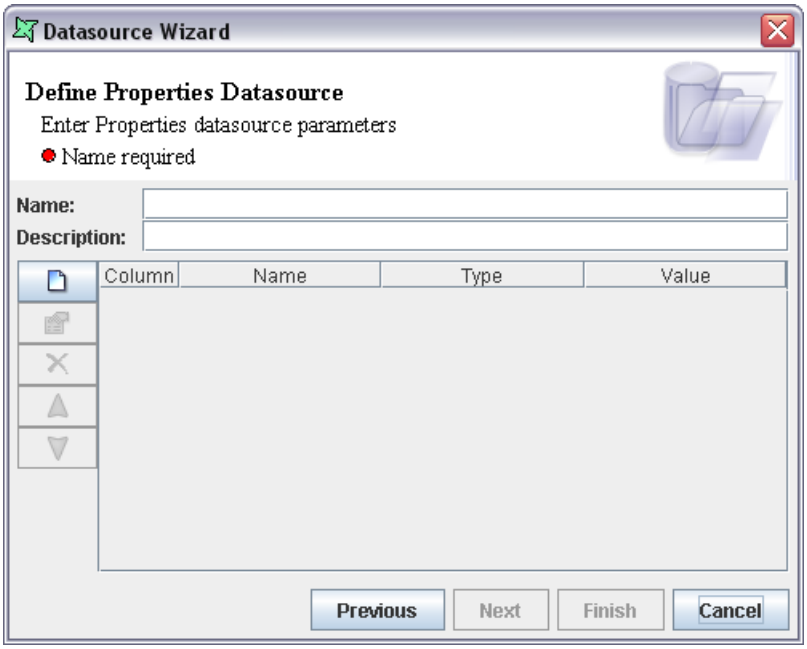
The Properties DataSource is used to provide a single record of data, usually either to a Parameters processor or it can be used as a direct data source.

The Properties DataSource can also be used to check the flow of data.

Properties DataSource Wizard

The Properties DataSource Wizard appears as shown in Figure 8.1, “Properties DataSource Wizard”.

Figure 8.1. Properties DataSource Wizard

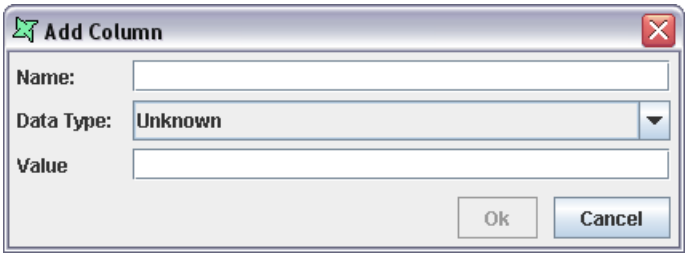


The name of the DataSource is entered in the Name text box. Any extra description that is used to describe the data source can be entered in the Description text box.

There are five buttons. They are Add Column, Edit Column, Move Up, Move Down and Remove Column buttons.

The Add Column button is used to add a new column. On clicking the Add Column button the Add Column dialog window pops up as shown in Figure 8.2, “Add Column”. The column name should be entered in the text box. The Data type should be selected from the combo box. The value is entered in the Value text box. On clicking the Ok button the column is added to the Wizard.

Figure 8.2. Add Column



The Edit Column, Move Up, Move Down and the Remove Column buttons are used to edit, reorder or delete columns.

Working with Properties DataSources

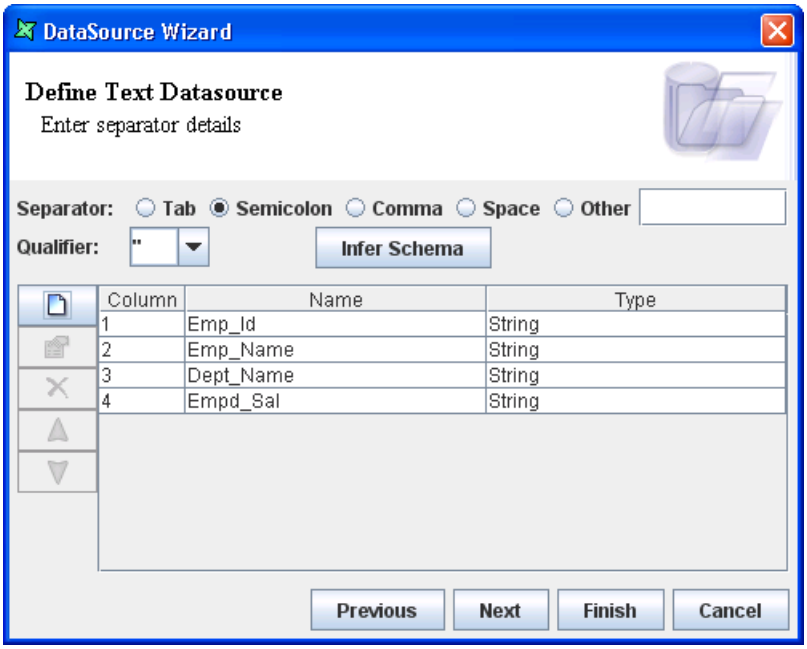
The Properties DataSource can be used directly or indirectly through a parameter processor. The examples given below illustrates the direct use of Properties DataSource.

Testing the Data Flow

The Employee details are maintained in a text file which includes the department field of the employees. Here's how to derive a new column using the department field of the text data source and check the validity of the derived column using the Properties DataSource:

1. Use the filesystem or folder popup menu to select Add -> DataSource.
2. The DataSource Wizard appears.
3. Select Text DataSource and click on the Next button. In the screen that appears enter a unique name for the data source, such as `Employee`.
4. Enter the URL in the text box or alternatively by clicking the button to the right of the text box select the EmpInfo.txt file from the Open dialog window. Select the "First line is header" check box.
5. Select the Qualifier as " and separator as Semicolon. On clicking the Infer Schema button the columns are inferred. After setting the properties the screen appears as shown in Figure 8.3, “Text DataSource Parameters”.Click the Finish button to add the text data source to the repository.

Figure 8.3. Text DataSource Parameters



6. Select the same location, and use the popup menu to select Add -> DataSource.

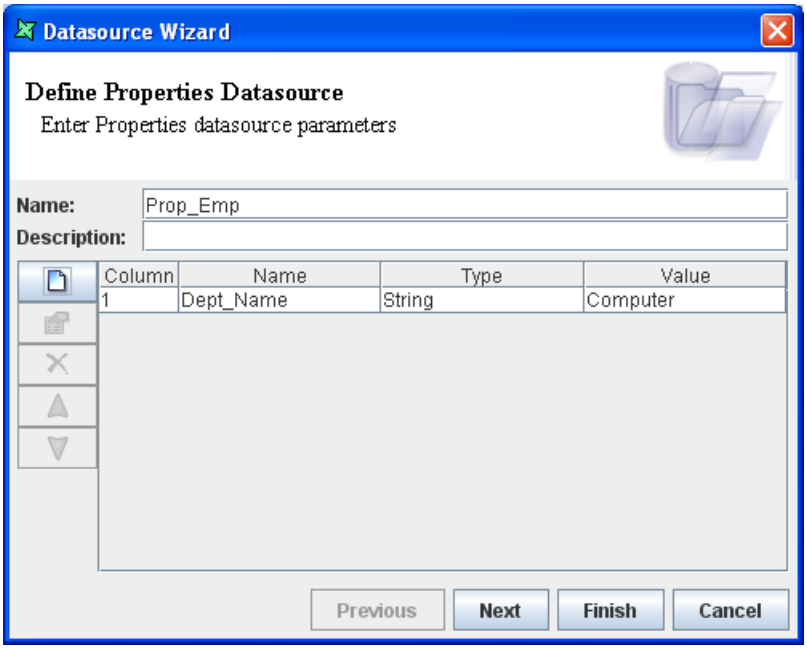
7. The DataSource Wizard appears. Select Properties DataSource and click on the Next button.

8. Enter a unique name such as `Prop_Emp`.

9. Click the Add Column button to invoke the Add Column dialog window. Enter column name as Dept_Name(same column name as in the text data source), Data type as String and the value as Computer. After entering the values the Add Column dialog window appears.

10. On clicking the Ok button the column is added to the Wizard. The screen appears as shown in Figure 8.4, “Properties DataSource”.

Figure 8.4. Properties DataSource



11. On clicking the Finish button the Prop_Emp data Source is added to the repository.

12. Select `Properties Samp`, use the popup menu to add a Composite DataSource called `Comp1`.

13. On selecting and double clicking on the `Comp1.ds` data source the composite designer window opens.

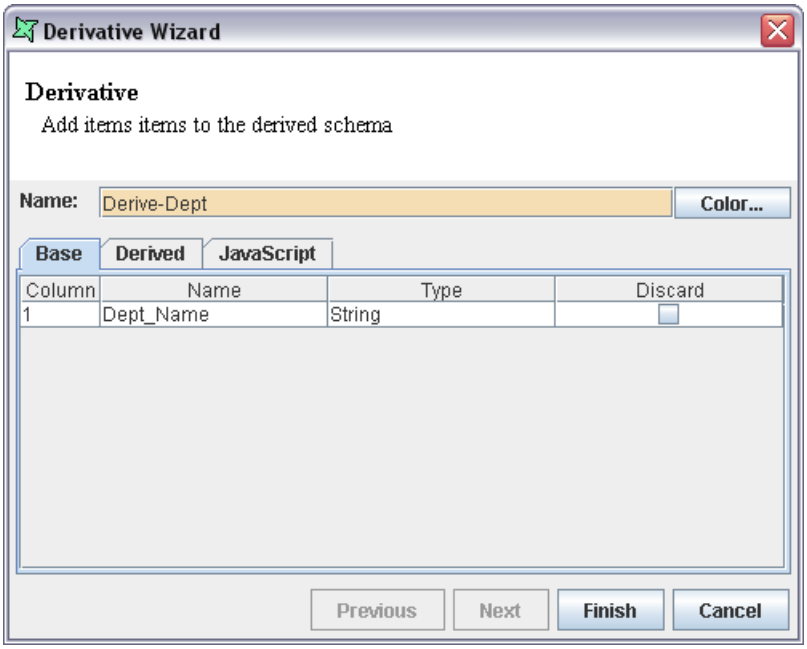
14. Select and place the properties data source Prop_Emp on the diagram. Select the derivative processor from the Action bar of the designer window and place it on the diagram.

15. Connect the output of `Prop_Emp` to the input of the Derivative processor.

16. Open the Derivative Properties and select the Derived tab and click on the Add Column button.

17. In the Add Column dialog, enter the name as Department, String as the data type and value as "dept". On clicking the Ok button the column is added to the wizard. After entering the properties the processor window appears as shown in Figure 8.5, “Derivative Wizard”.

Figure 8.5. Derivative Wizard



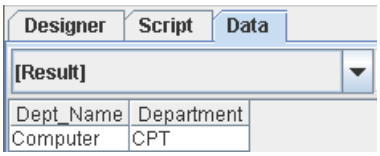
18. Select the JavaScript tab and enter the following script in the window.

```
if (Dept_Name== "Computer") dept = "CPT";
else dept = "Others";
```

19. Click the Finish button.

20. Connect the output of the `Derive-Dept` processor to the Result. Select Result and choose View from the popup menu. The output will be displayed in the data window as shown in Figure 8.6, “Test Output”. It can be seen that the department Computer is mapped to CPT.

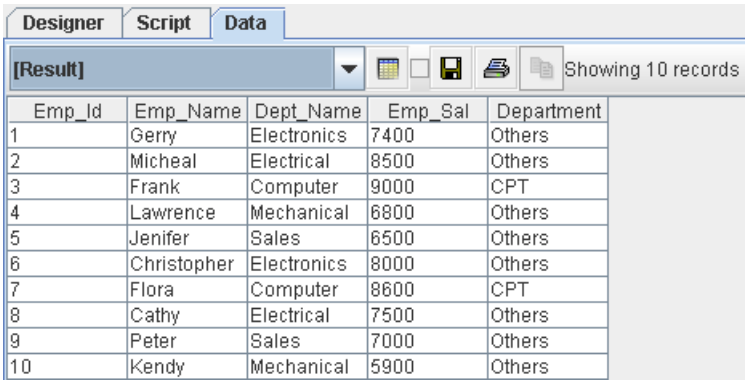
Figure 8.6. Test Output



21. Now delete the flow between the Prop_Emp data source and the Derive-Dept processor.

22. Select and place the Employee.ds data source in the designer window. Using the flow processor connect the output of the Employee.ds data source to the input of the Derive-Dept processor. Select Result, and choose View from the popup menu. The output will be displayed in the data window as shown in Figure 8.7, “Full Output”. It can be seen all the records containing the Computer department are to "CPT" and the Other departments are mapped to others. Thus using the above procedure it becomes easier to check the validity of the derived columns.

Figure 8.7. Full Output



Emp_Id	Emp_Name	Dept_Name	Emp_Sal	Department
1	Gerry	Electronics	7400	Others
2	Micheal	Electrical	8500	Others
3	Frank	Computer	9000	CPT
4	Lawrence	Mechanical	6800	Others
5	Jenifer	Sales	6500	Others
6	Christopher	Electronics	8000	Others
7	Flora	Computer	8600	CPT
8	Cathy	Electrical	7500	Others
9	Peter	Sales	7000	Others
10	Kendy	Mechanical	5900	Others

Note

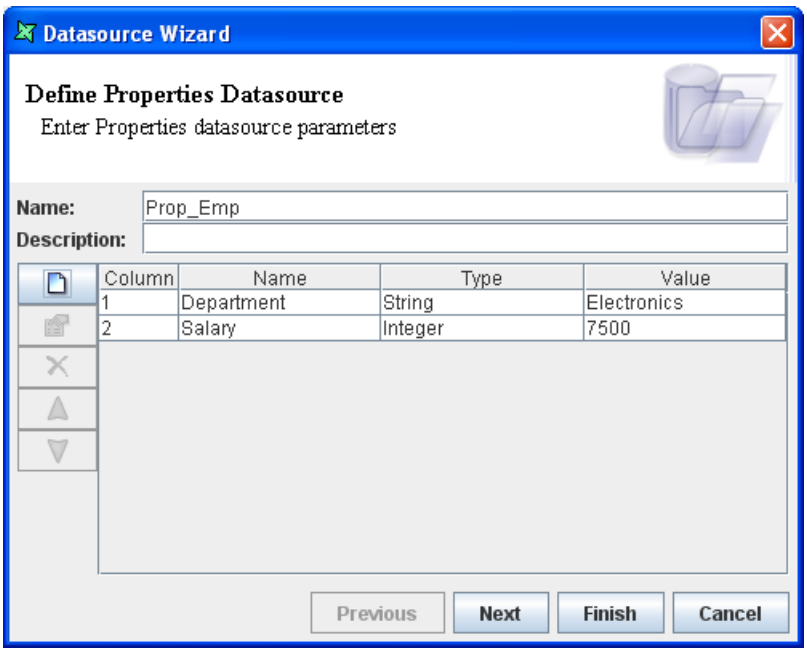
Therefore, if you want to test a flow with specific values, you can use a Properties data source to provide them, then switch back to the real data source for release.

Passing Parameters to the Flow

Here's how to use a Properties DataSource along with the Parameter processor to pass parameter values for checking a Filter condition:

1. Add a Properties DataSource named `Prop_Emp1`.
2. Add a column called Department with type String and value Electronics.
3. Add a column called Salary with type Integer and value 7500. The values are shown in Figure 8.8, “Prop_Emp1”.

Figure 8.8. Prop_Emp1



4. Click the Finish button to add the data source to the repository.

5. Add a Composite DataSource called `Comp2`.

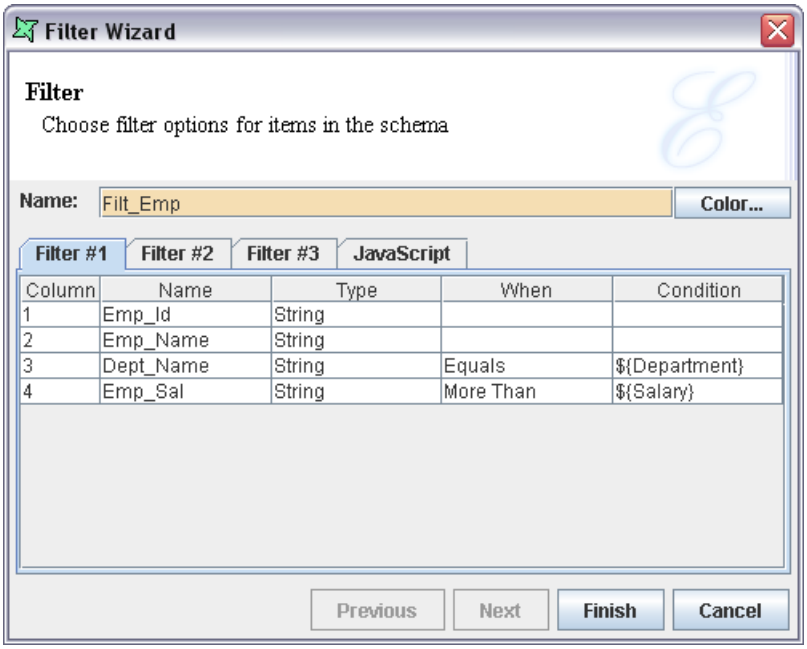
6. Open the Composite diagram and and drag the Employee.ds onto it. Add a Filter processor to the diagram as well.

7. Connect the output of the Employee DataSource to the input of the Filter processor.

8. Open the filter properties. In the row corresponding to Dept_Name field select "Equals" option from the combo box of When column. Enter the condition as ${Department}.

9. In the row corresponding to Emp_Sal field select "More Than" option from the combo box of When column. Enter condition as ${Salary}. After entering the values the page appears as shown in Figure 8.9, “Filter Wizard”. Click the Finish button.

Figure 8.9. Filter Wizard



10. Select the Parameters processor from the action bar and place it in the workspace of the designer window. Change the colour of the processor.

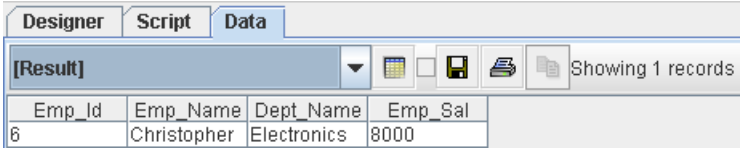
11. Connect the output of the Filter processor to the input of the Parameters processor

12. Select and place the Prop_Emp1 data source in the designer window workspace.

13. Connect the Prop_Emp1 DataSource with the params input of the Parameters processor and the output of the Parameters processor to the Result.

14. Select Result, and choose View from the popup menu. The output is displayed as shown in Figure 8.10, “Output”.

Figure 8.10. Output



15. The values passed by the records of the Prop_Emp1 data source are matched with the dynamic parameters in the filter condition. Based on this the records of the text data source satisying the filter condition are fetched.

As the flows are pull-push, the data store pulls data from the source, this is when the parameters get added. The pull command propogates back to the filter which uses the properties to control the filter process. The pull command terminates at the datasource, which pushes the data back along the flow, through the filter which has already been configured.

Chapter 9
Reference DataSource

Overview

The Reference data source acts as a reference to another data source and supplies parameters to it. In other words, a Reference DataSource is a proxy data source with preset parameter values.

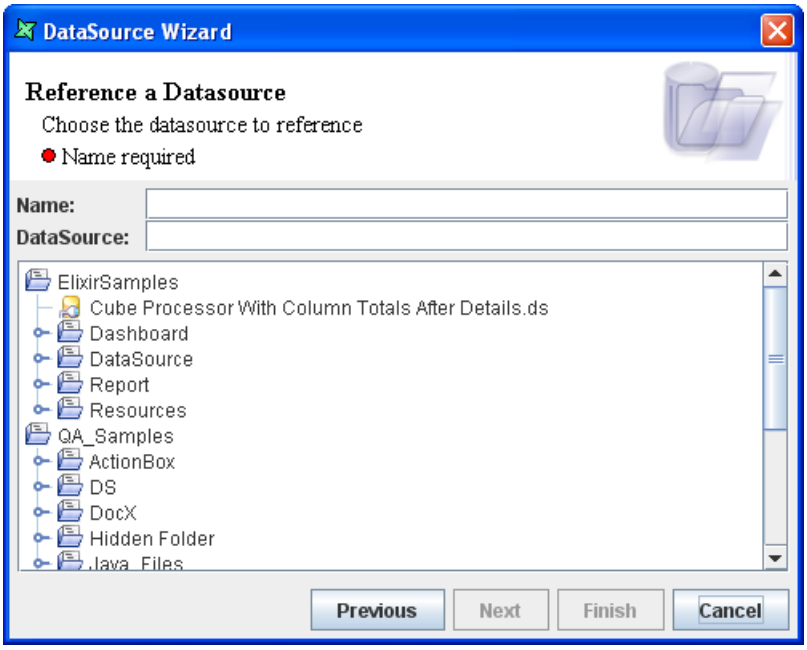
The Reference DataSource points to a database and "personalizes" it for each user environment, for example supplying different queries or user id and password. The difference between a Reference DataSources and Dynamic parameters, is that the user doesn't need to see any UI, or explicitly enter any values each time the DataSource is loaded. The data source designer has done all the work.

A Reference DataSource can be used to wrap any kind of DataSource (including a Composite or even another Reference DataSource) to fix certain parameters without the need for a complex Composite + Properties + Parameter Processor combination.

Reference DataSource Wizard

The Reference DataSource Wizard is shown in Figure 9.1, “Reference DataSource Wizard”.

Figure 9.1. Reference DataSource Wizard

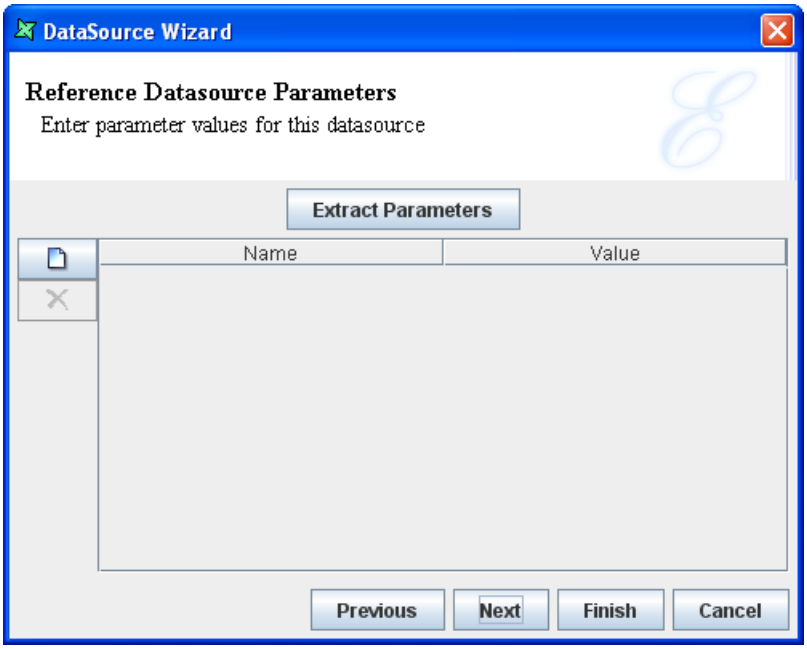


Name: Enter a unique name for the Reference DataSource.

DataSource: Select the DataSource for which the parameter values need to be set.

Click on `Extract Parameters` seen in Figure 9.2, “Reference DataSource Wizard Page Two” to extract parameters from the datasource or define parameters explicitly using buttons on the left. Click on `End` to end the creation process.

Figure 9.2. Reference DataSource Wizard Page Two



Working with Reference DataSource

The different ways of using a Reference DataSource are given below:

Wrapping an Excel DataSource

In this illustration an Excel DataSource has been added in which a parameter has been specified for the range and the values are expected to be entered during the loading of the Excel file.

Here's how to add a Reference DataSource and use it to supply values automatically to the parameterized Excel data source.

The sample file Empdata.xls consists of employee details in two worksheets as shown in Figure 9.3, “Empdata.xls”.

Figure 9.3. Empdata.xls

	A	B	C	D
1	**Emp_Id**	**Emp_Name**	**Dept_Name**	**Emp_Sal**
2	1	Gerry	Electronics	7400
3	2	Micheal	Electrical	8500
4	3	Frank	Computer	9000
5	4	Lawrence	Mechanical	6800
6	5	Jenifer	Sales	6500
7	6	Christopher	Electronics	8000
8	7	Flora	Computer	8600
9	8	Cathy	Electrical	7500
10	9	Peter	Sales	7000
11	10	Kendy	Mechanical	5900
12				

Sheet1 Sheet2

1. Before adding an Excel DataSource, the ranges must be defined in the Excel file. Open the Empdata.xls file with Microsoft Excel and select Name -> Define under the Insert menu. The Define Name dialog box pops up.

Enter name as Emp1_All and enter range as given below in the "Refers to:" text box and click the Add button.

```
=Sheet1!A$1:D$11
```

Enter name as Emp2_All and the range as given below in the "Refers to:" text box and click the Add button.

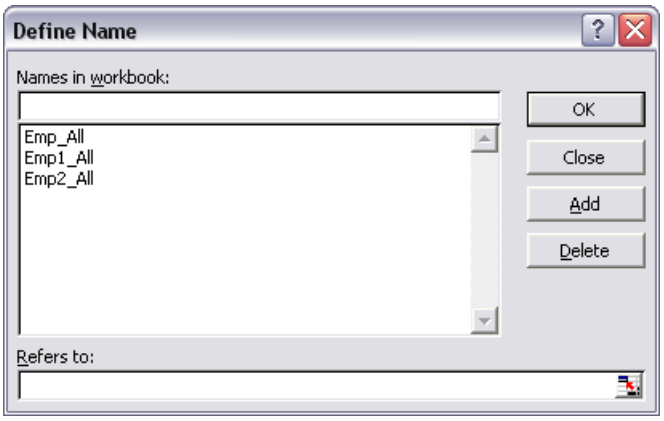
```
=Sheet2!A$1:D$6
```

Enter name as Emp_All and the range as given below in the "Refers to:" text box and click the Add button.

```
=Sheet1:Sheet2!A$1:D$11
```

After adding the columns the dialog window appears as shown in Figure 9.4, “Define Name”. Click the Ok button.

Figure 9.4. Define Name

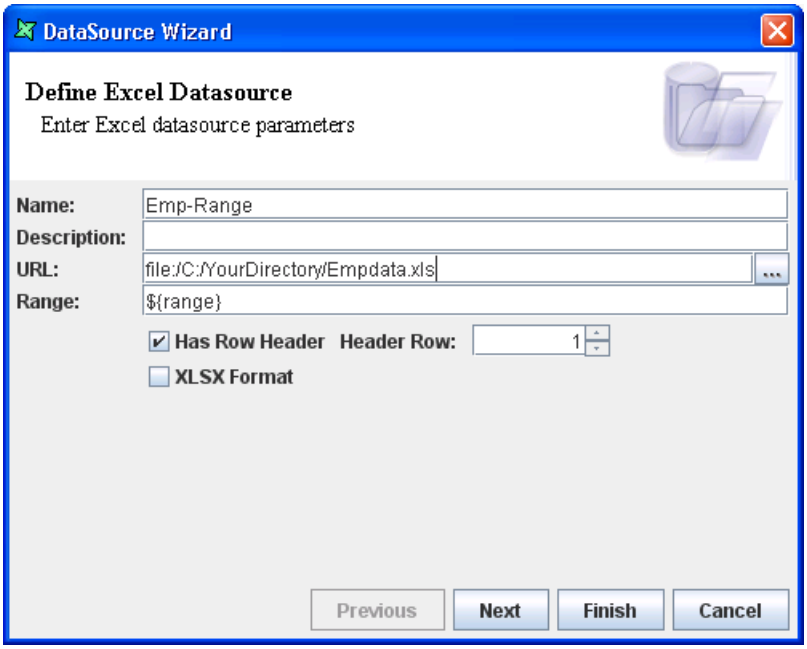


Save the Empdata.xls file.

2. Now move to Elixir Repertoire and create a new Excel DataSource called `Emp-Range`. Enter the `Empdata` URL in text field provided or click the button to the right of the text field, to select the Empdata.xls file from the Open dialog window.

   Enter the Range as ${Range} and select the First Row Header check box. After setting the properties the DataSource Wizard appears as shown in Figure 9.5, “Excel DataSource Sample”. Now click on the Next button. In the screen that appears click on the Infer Schema button. Enter any range specified above in the Dynamic Parameters dialog box that appears and click the Finish button. The schema will be inferred. Change the Data Type of Emp_Id and Emp_Sal to Integer. Finally, click the Finish button.

Figure 9.5. Excel DataSource Sample

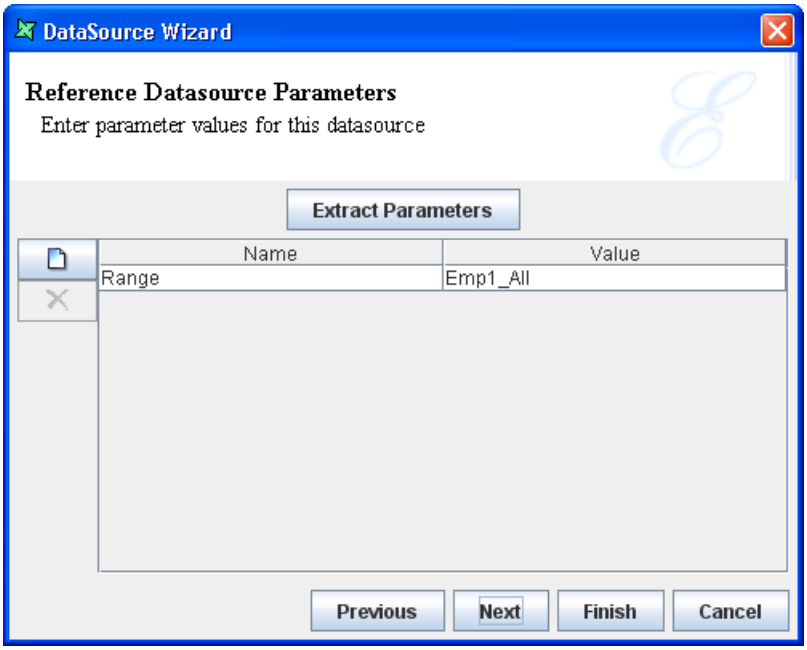


Click the Finish button to add `Emp-Range.ds` to the repository.

3. Choose Add -> DataSource again and this time select Reference DataSource and click the Next button.

   In the "Reference a DataSource" screen that appears enter Name as `Ref-Ex1` and select the `Emp-Range` from the DataSource combo box. Enter Emp1_All as the value for Range. After entering the values the screen appears as shown in Figure 9.6, “Reference a DataSource”. On clicking the Finish button, the Ref-Ex1.ds is added to the repository.

Figure 9.6. Reference a DataSource

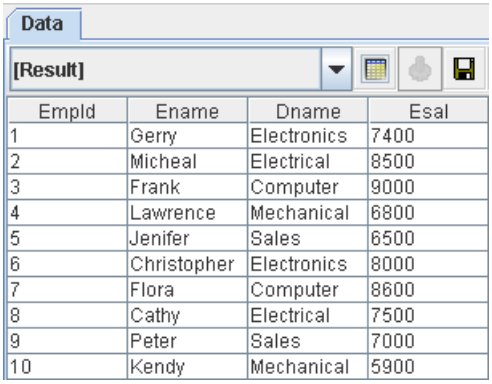


4. Similarly, add another Reference Datasource called `Ref-Ex2` which also references `Emp-Range`. Enter Emp2_All in the Range text field. On clicking the Finish button, the Ref-Ex2.ds is added to the repository.

5. Similarly, add a Reference DataSource called `Ref-Ex3` as above but enter `Emp_All` in the Range text field.

6. Now we have three sample Reference DataSources we can look at how they work. Open the Ref-Ex1 data source. In the data window click on the Load Data menu. The output is displayed as shown in Figure 9.7, “Sample Output”. It is seen that the data from the first sheet of the Excel file is displayed.

Figure 9.7. Sample Output



EmpId	Ename	Dname	Esal
1	Gerry	Electronics	7400
2	Micheal	Electrical	8500
3	Frank	Computer	9000
4	Lawrence	Mechanical	6800
5	Jenifer	Sales	6500
6	Christopher	Electronics	8000
7	Flora	Computer	8600
8	Cathy	Electrical	7500
9	Peter	Sales	7000
10	Kendy	Mechanical	5900

7. Opening Ref-Ex2 and loading the data shows the data from the second sheet of the Excel file is displayed

8. Repeating the same action on Ref-Ex3 shows the data from both the worksheets of the Excel file is displayed.

Thus all the three Reference DataSources refer to the same Excel file, but parameterize it in different ways. This means there is only one place to define common information, such as the Excel file location, making the solution easier to maintain.

Wrapping a Composite DataSource

In this example we will work with a text file containing employee details and filter the records with a Composite wrapped with a Reference Datasource.

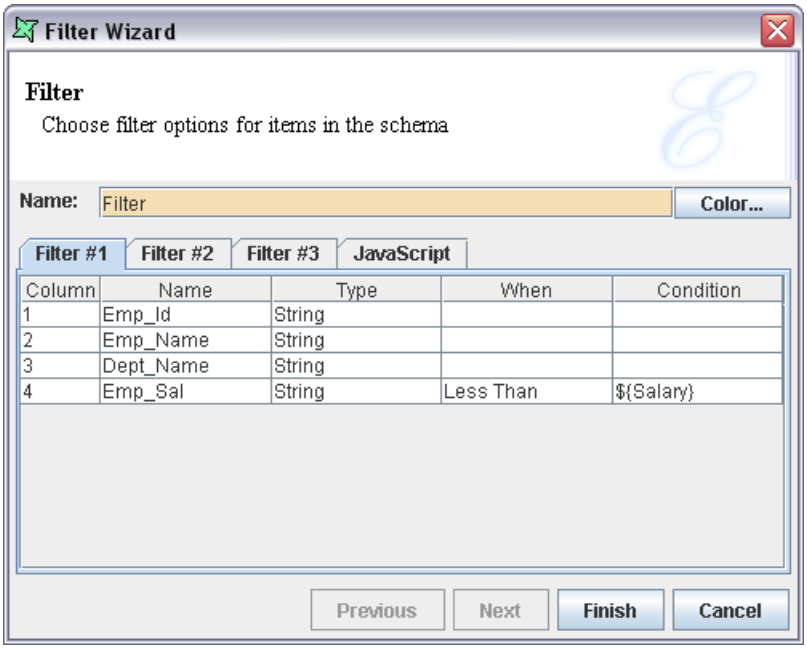
1. Add a new Text DataSource called Employee and set the URL to reference the sample Empinfo.txt. Select "First line is header option" check box and select the Access type as Separator Character. Click the next button.

   In the screen that appears enter Qualifier as ". Select Semicolon option as the Separator. Click the Infer Schema button. The fields of the text data source are inferred. On clicking the Finish button the text data source is added to the repository.

2. Now add a Composite DataSource called `Compo`. Drag and drop the Employee.ds from the Repository over the Composite diagram. Add a Filter processor to the diagram and connect it to the Employee datasource.

   Open the Filter properties and select Filter#1 tab window. In the row corresponding to Emp_Sal field select "Less Than" option from the combo box of When column. Enter condition as ${Salary}. The screen appears as shown in Figure 9.8, “Filter Condition”. Click the Finish button.

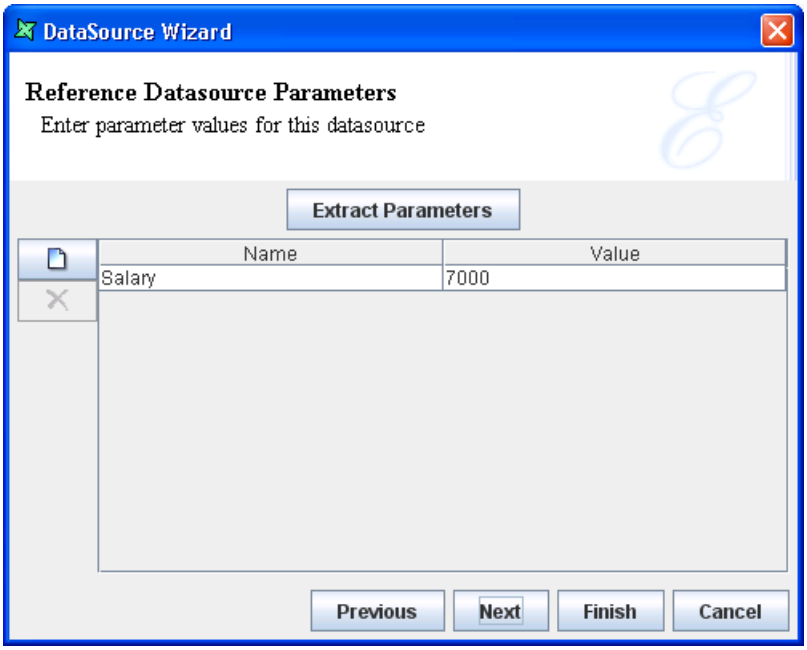
Figure 9.8. Filter Condition



Connect the output of the Filter processor to the input of the Result.

3. Now add a Reference DataSource named `Ref-Com1`. Select the `Compo` data source from the DataSource combo box. Enter 7000 as the value for the Salary parameter. After entering the values the screen appears as shown in Figure 9.9, “Sample Reference”. On clicking the Finish button, the Ref-Com1.ds is added to the repository.

Figure 9.9. Sample Reference



4. Repeat the process to create `Ref-Com2` and enter 8000 as the Salary value.

5. Open Ref-Com1.ds and confirm that only the records of employees whose salary is less than 7000 are displayed.

6. Similarly, open Ref-Com2.ds and check that only the records of employees whose salary is less than 8000 are displayed.

Thus both the Reference DataSources refer to the same Composite DataSource, but parameterize it in different ways.

Chapter 10
Object DataSource

Overview

The Object DataSource allows data to be extracted at runtime from Java objects and Enterprise Java Beans. If your Java classes comply with standard bean accessor patterns (eg. getXXX and isYYY), the values can be easily extracted. JavaScript is used to coordinate the data access so that it can be maintained within the interactive Elixir Data Designer environment and doesn't need a separate compile cycle.

The Classpath

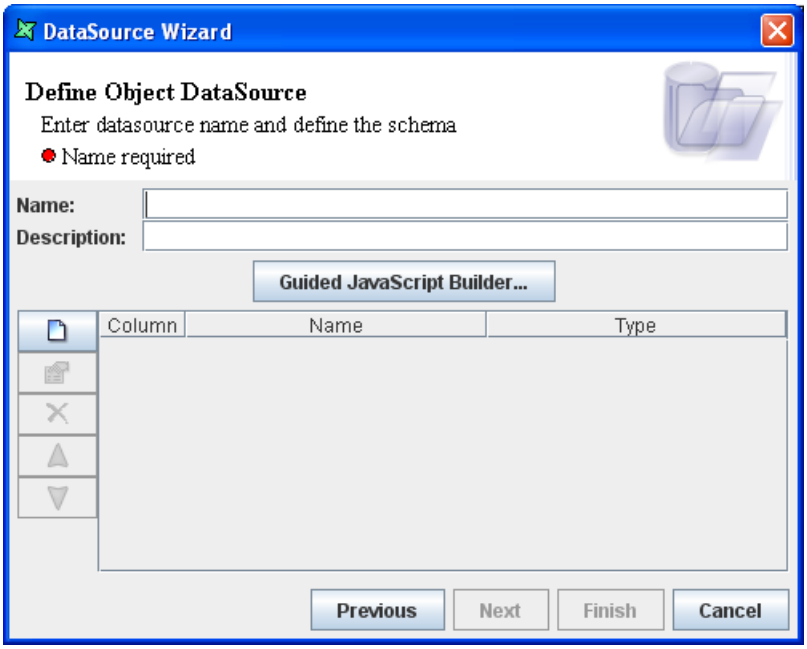
In order to access user-defined classes, they must be packaged as a jar and placed with any dependent jars in the `ext` directory of your installation. This directory is scanned when the program starts, so you will need to start the program after adding the jars to the appropriate location.

In the stand-alone Repertoire Designer, the `ext` location is the directory called ext within the Repertoire install directory. On the server, or when running Remote, the `ext` location is the directory called ext within the RepertoireServer install directory. Note that when running Remote, the ext jars remain on the server, but class and method names from the jars are provided to the Remote tool to allow browsing of the classes within the Guided JavaScript Builder in the Remote GUI which is described below.

Object DataSource Wizard

The First Screen of the Object DataSource Wizard appears as shown in Figure 10.1, “Object DataSource Wizard” In this screen the Object DataSource name and schema have to be defined.

Figure 10.1. Object DataSource Wizard



The Name of the data source must be unique. Any extra description that is used to describe the data source can be entered in the Description text box.

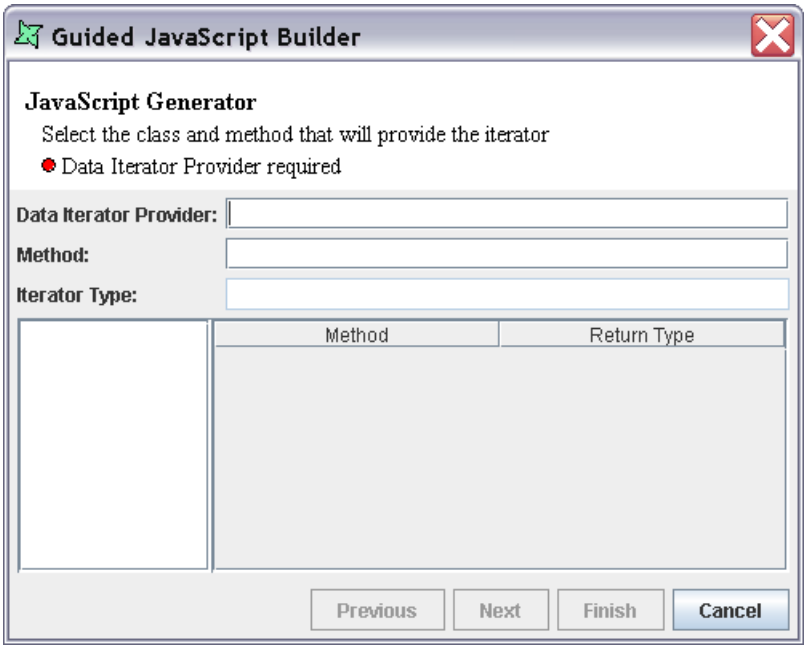
On this page you can either infer or edit the data class schema using one of two methods - either enter the schema and JavaScript manually (if you are an experienced Java developer) or use the Guided JavaScript Builder.

Guided JavaScript Builder

With the Guided JavaScript Builder button you are assisted by a wizard that helps you to automatically define the schema and the JavaScript. Of course, you can edit these manually later, so this approach is a useful first step toward customizing the data interface.

On clicking the Guided JavaScript Builder the screen appears as shown in Figure 10.2, “Guided JavaScript Builder”. In this screen the details of the Data provider Iterator class is entered. The tree view showing the classes available in the `ext` jars is displayed on the left side of the screen. Select the Data Iterator provider class. The Methods and the corresponding Return Type of the class are listed in a table on the right. On selecting the Iterator method from the list of available methods the method name and the Iterator type is displayed in the corresponding text fields.

Figure 10.2. Guided JavaScript Builder



Click on the Next button. In this screen the data provider class of the Object DataSource is selected. On selecting the Data Class from the tree, the class file name is automatically entered in the Data Class text field.

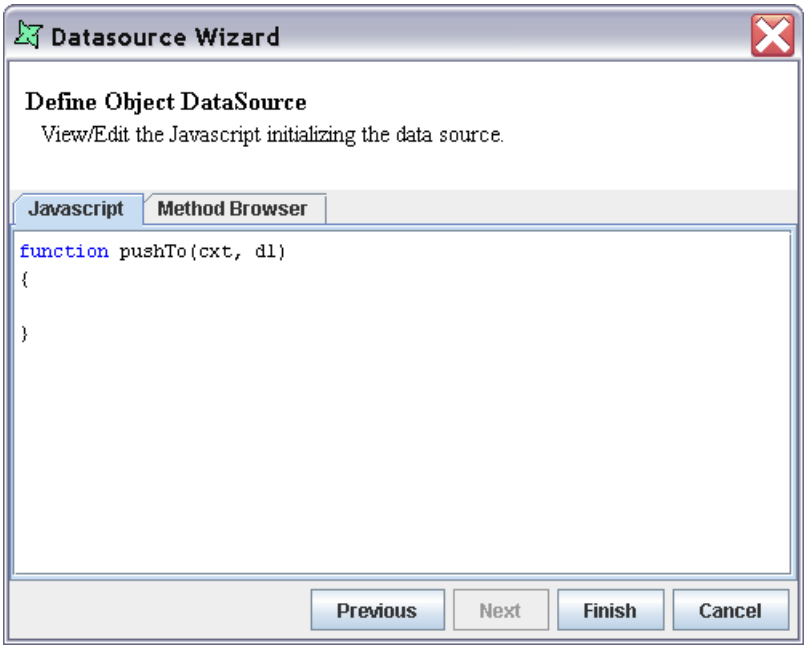
Finally, on clicking the Finish button the fields names and data type of the selected class are automatically added to the schema definition table

By clicking the Next button, you can see the JavaScript that has been auto-generated.

Manual Creation

By clicking the Add Column button you can enter the Data class field name and select the corresponding data type from the combo box in the Add Column dialog window. On clicking the OK button the column will be added to the Schema Definition table. You then proceed onto the next screen Figure 10.3, “JavaScript Editor”. Here you can manually enter the JavaScript code that contains the Iterator method to access the fields in the data source. The Method Browser window lists all the methods and return types from which the methods to be included in the code can be selected and included manually in the JavaScript code.

Figure 10.3. JavaScript Editor



The JavaScript entered here must conform to the following pattern:

```
function pushTo(/*PushContext*/ cxt, /*DataListener*/ dl)
{
  // put any imports here

  // call start data to begin rendering the header
  dl.startData(this);

  // get the data to be iterated over
  var dIter = Company.getEmployeesAsIterator();
  while (dIter.hasNext())
  {
    // get the data values to supply
    dataObj = dIter.next();

    // create a record instance to supply the values in
    rec = this.newRecordInstance();

    // get the Object[] to hold the values
    data = rec.getData();

    // fill in the values according to the schema
    data[0] = dataObj.getDateJoined();
    data[1] = dataObj.getDeparment();
    // etc.

    // call process record to begin rendering a detail
    dl.processRecord(rec);
  }

  // call end data to begin rendering the footer
  dl.endData(this);
}
```

You can use the JavaScript Builder to generate a first-cut and edit it if you prefer. Remember that each time you use the JavaScript Builder it will rewrite the JavaScript, so you will lose any changes you have made. Finally, on clicking the Finish button the Object Data Source will be added to the repository.

Object DataSource API

The pushTo method is the engine that drives all data source record generation. It takes two parameters, `cxt`, a PushContext object, and `dl` a DataListener object. This section will describe the available methods provided by these two objects and the supporting objects that they depend on.

PushContext

The PushContext provides useful methods for other data sources, but is unlikely to be useful within JavaScript as the functions it provides can be done directly.

String getParameter(String name)	Get the value of a parameter. You should use ${substitution} in JavaScript instead of calling this method, as substitutions are detected and will be included in prompts, whereas calls to getParameter will not.
String substitute(String s)	Performs ${substitution} replacement on an input string. Again, this is not useful within JavaScript, as you can use ${substitution} strings directly in the code.

DataListener

A DataListener receives data from a DataSource for subsequent processing. For Java use, DataListener (an interface) and all the classes referenced by it are in the com.elixirtech.data2 package. The methods listed are all public.

void startData(IDataSource src)	This method must be invoked before sending any records or groups, to allow the listener to prepare for receipt. The `src` should be `this` (the ObjectDataSource itself).
void startGroup(DataGroup group)	This method is called to indicate that the subsequent records are part of a group. This method should not be called if the data is not already sorted and grouped. Where data is grouped, groups may be nested, within other groups, but all records must be within the innermost groups. You are not allowed to have a group that contains both records and child groups. Similarly, if the top level is grouped, it can contain no records outside of those groups.
boolean processRecord(DataRecord record)	Each record that the data source supplies is passed to the listener through this method call. Usually the method will return true. If the method returns false, then it indicates that the listener does not want to receive any more records. In this case, you can choose to stop sending records (additional ones will just be discarded anyway), but must send the necessary endGroup and endData calls to gracefully terminate the operation. The same applies if exceptions are caught - you should still send the necessary symmetrical endXXX calls to balance the startXXX calls you have already made.
void endGroup(DataGroup group)	This method is called to indicate the end of a group. Each call to startGroup should be matched with a corresponding call to endGroup after the necessary records have been processed.

void endData(IDataSource src)	This method must be invoked after sending all records and groups to indicate that no more information is available. The `src` should be `this`.

DataGroup

DataGroups delimits a set of sorted records into groups.

DataGroup(int level, String name)	Construct a DataGroup at a particular level (starting at one). If groups are nested, then child groups would be at level two, etc. The name may be any String - the use depends on the DataListener. For example, an Excel DataStore will use the group name as the name of the Sheet.
int getLevel()	Retrieve the group level
int getName()	Retrieve the group name

DataRecord

A DataRecord supplies an array of objects conforming to the DataSource schema to the DataListener. The ObjectDataSource supplies a method `newRecordInstance()` to instantiate a record with the appropriate structure. Note that records cannot be reused - once a record has been passed to the DataListener, you cannot modify it and pass it again.

Object[] getData()	Get the array backing this DataRecord so that you can set values before passing it to the DataListener. You should set the values in the array according to their order in the schema (zero-based).

Working with Object DataSources

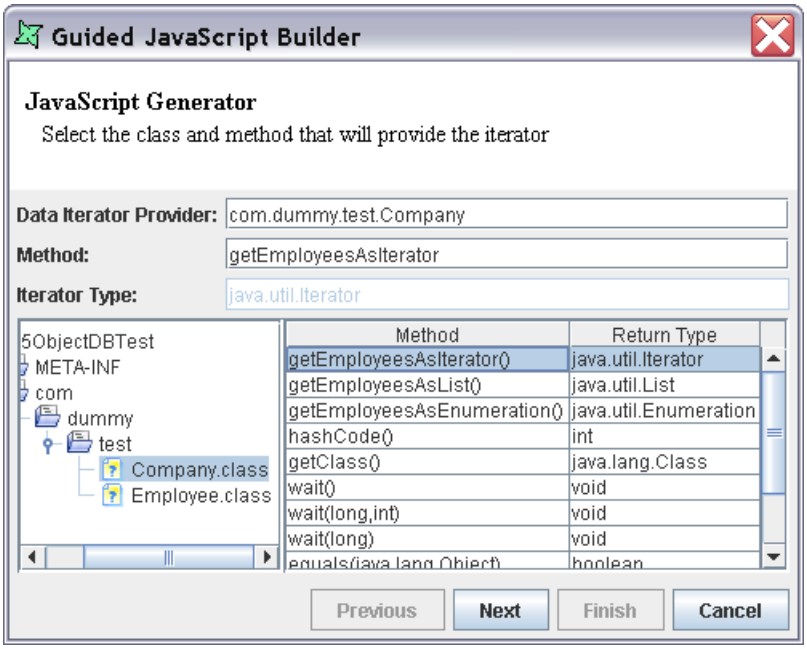
There are two files namely the "Company.java" and the "Employee.java" file. Here's how these files are to be used for providing data and the Iterator method while adding the Object Data Source.

1. Review the sample files Employee.java and Company.java. Both the files need to be compiled and jarred (The jar has been created for you called ObjectDB.jar). Remember the location of the jar file as we will need to add it to the class path in the Object DataSource.

2. Create a new Object DataSource called `Object1` and enter the classpath for ObjectDB.jar.

3. Click the Next button and choose the Guided JavaScript Builder. On the builder select `com.dummy.test.Company` from the tree as the Data Iterator provider class. The methods associated with this class are listed in the table on the right. Select getEmployeesAsList() method from the list of available methods. The method and the corresponding Iterator type is displayed in the text fields. After selecting the method and Iterator type the screen appears as shown in Figure 10.4, “Sample Data Iterator”.

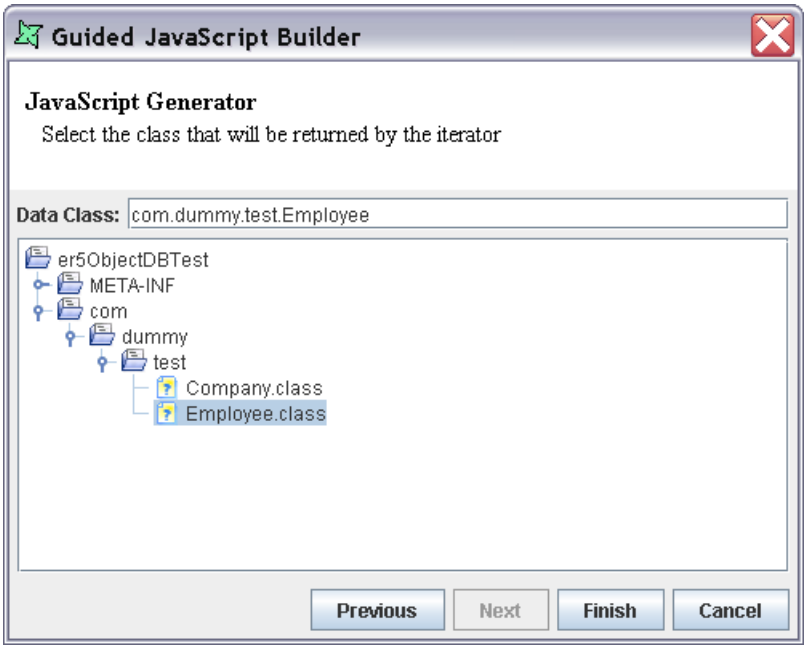
Figure 10.4. Sample Data Iterator



4. Click on the Next button. The classpath element is displayed as a tree view in the window.

5. Expand the tree view and select the "Employee.class" file which is the data provider class. The selected Data Class name is automatically displayed in the Data Class text box. After entering the class details the screen appears as shown in Figure 10.5, “Sample Data Class”.

Figure 10.5. Sample Data Class



6. Click the Finish button. The Data Class fields will be automatically added to the Schema Definition table.

7. The Next screen shows the JavaScript code that has been generated.

```
function pushTo(cxt,dl)
{
  dl.startData(this);
  importClass(Packages.com.dummy.test.Company);
  var dIter = Company.getEmployeesAsIterator();
  while (dIter.hasNext())
  {
    rec = this.newRecordInstance();
    data = rec.getData();
    dataObj = dIter.next();
    data[0] = dataObj.getDateJoined();
    data[1] = dataObj.getDeparment();
    data[2] = dataObj.getDesignation();
    data[3] = dataObj.getName();
    data[4] = dataObj.getID();
    data[5] = dataObj.hashCode();
    data[6] = dataObj.getClass();
    data[7] = dataObj.toString();
    dl.processRecord(rec);
  }
  dl.endData(this);
}
```

8. On clicking the Finish button the Object data source is added to the repository.

9. View the data to verify the correct data is returned.

Chapter 11
LDAP DataSource

Overview

LDAP is a Lightweight Directory Access Protocol. It is a protocol for accessing information directories such as organizations, individuals, phone numbers, and addresses. LDAP servers store data in a Directory Information Tree, which is a hierarchical grouping of related data.

The LDAP Data source driver is built on JNDI API to provide access to the directories. It provides the user the ability to filter and extract LDAP entries.

LDAP is based on the X.500 standard, but is significantly simpler. Unlike X.500, LDAP supports TCP/IP, which is necessary for any type of Internet access. A typical LDAP server is a simple network-accessible database where an organization stores information about its authorized users and what privileges each user has. The standards are specified in RFC 1777.

LDAP DataSource Wizard

The LDAP DataSource Wizard is shown in Figure 11.1, “LDAP DataSource Wizard”.

Name: Enter the DataSource name in the text box. This should be a unique name.

Description: Any extra description that is used to describe the data source can be entered in the Description text box.

Host: The host name or IP address of the server is specified here.

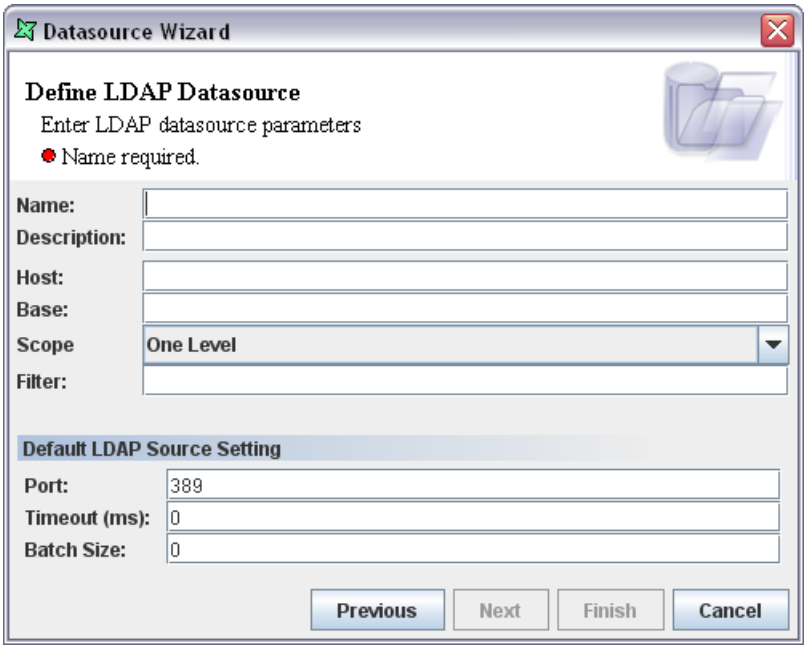
Base: The Base, sometimes called Distinguished Name (DN) is used to uniquely identify entries in LDAP. The base is always a fully qualified name that identifies entries starting from the root of the LDAP name space (as defined by the server).

The following string-type attributes represent the set of standardized attribute types for accessing an LDAP directory. The Base can be composed of attributes using the LDAP syntax. For example:

- CN - CommonName
- L - LocalityName
- O - OrganizationName
- OU - OrganizationalUnitName
- C - CountryName
- STREET - StreetAddress

If there are domain components `com` and `example`, then the base for the com.example domain is "dc=com,dc=example". Similarly, if the organizational unit comes under the domain com.example, then to access people within the organization you would use "ou=people,dc=example,dc=com".

Figure 11.1. LDAP DataSource Wizard



Scope: Scope defines the set of information used to search for data. You may choose one of three values: Object, One Level or Subtree. Choosing Object will only return data held by the object identified by the Base. Choosing One Level will return data held by children of the object identified by the Base (this is the default). Finally Subtree will return data from all children in the subtree below the Base.

Filter: Search filters enable you to define criteria for more efficient and effective searches. LDAP filters are used to specify criteria for directory search. Filters are optional - if you leave it blank a default filter "objectclass=*" will be used.

In filter, the matchingAttrs argument is converted into a string filter that is a conjunctive expression of the attributes from matchingAttrs.

For example, when a matchingAttrs containing the attributes sn:Geisel and mail: (no value given), it is translated into the string filter "(&(sn=Geisel)(mail=*))".

The filter conventions are given in the ftp://ftp.isi.edu/in-notes/rfc2254.txt link. [ftp://ftp.isi.edu/in-notes/rfc2254.txt link]

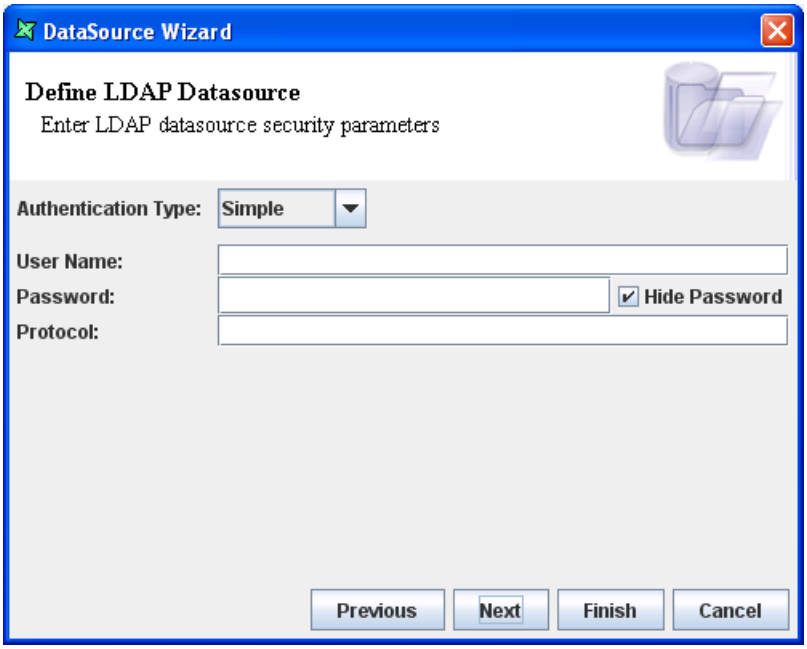
Port: The default port for connection is 389. The LDAP port number of the machine on which the Directory Services is running is specified in the text box.

Timeout: This is the time interval allowed for certain operation to occur. The LDAP timeout is the time limit required to wait for the result. If 0 is specified then it means that the datasource will wait indefinitely.

Batch Size: The value of the batch size property specifies the batch size of the search results returned by the server. The batch size is specified in the Batch Size text box. A setting of 0 means that the provider should block until all the results have been received. If for instance the batch size is specified as 24 then the provider should block until 24 entries have been read from the server or until the enumeration terminates, whichever produces fewer number of results.

On clicking the Next button, the LDAP Security page will appear. In Figure 11.2, “Security Parameters”, the default type "none" has been changed to "simple" so that you can see the available options. If your server doesn't require any authentication for read access, you can leave the authentication type as none.

Figure 11.2. Security Parameters



Authentication Type: The value of this property is a string that specifies the authentication mechanism(s) for the provider to use. The following values are defined for this property:

1. none: If this option is selected from the combo box, then no authentication is required. It is referred to as anonymous bind.

2. simple: If this option is selected then a clear text password is used. This type of bind sends an unencrypted user name and password to the LDAP server for verification, and can be secured only by using a secure channel to transmit the password, eg. SSL.

3. GSSAPI: GSSAPI stands for Generic Security Services Application Programming Interface. An application level interface (API) to system security services. It provides a generic interface to services which may be provided by a variety of different security mechanisms.

4. Digest-MD5: In Digest-MD5, the LDAP server sends data that includes various authentication options that it is willing to support plus a special token to the LDAP client. The client responds by sending an encrypted response that indicates the authentication options that it has selected. The response is encrypted in such a way that proves that the client knows its password. The LDAP server then decrypts and verifies the client's response.

User Name: This property is used to specify the identity of the principal for authenticating the caller to the service.

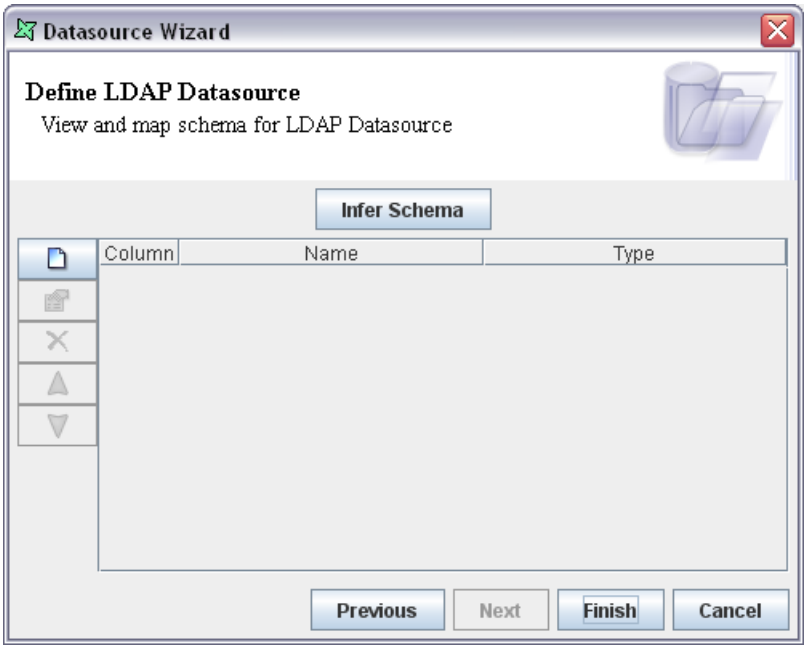
Password: This property is used to specify the credentials of the principal for authenticating the caller to the service. By default, any text entered here is shown as asterisks (*), unless you turn off "Hide Password".

Protocol: The value of this property is a string that specifies the security protocol for the provider to use. The protocol is entered in the text box.

When you click on the Next button the screen appears as shown in Figure 11.3, “LDAP DataSource Schema”.

On clicking the Infer Schema button the settings you have entered so far are used to query the LDAP server and extract the schema attributes. The fields and the corresponding data types are displayed in

Figure 11.3. LDAP DataSource Schema



the table. On clicking the Finish button in the data source Wizard the data source will be added to the repository.

Working with LDAP DataSource

Here's how to access the organizational unit ou=people of the DNS domain example.com.

The steps given below are followed to add the LDAP data source and list the people:

1. From a filesystem or folder popup menu, choose Add a Datasource, select LDAP DataSource types and click the Next button.
2. In the screen that appears, enter the name of the DataSource as Organization.
3. Enter the local host name or the IP address of the client system in the Host text field.
4. Enter "ou=people, dc=example, dc=com" in the Base text field.
5. Enter the port number of the server. After setting the properties the screen appears as shown in Figure 11.4, “LDAP Datasource Wizard”. Click the Next button.
6. In the screen that appears select 'none' option from the Authentication Type combo box. Click the Next button.
7. In the screen that appears click the Infer Schema button. The attribute names and types are listed in the table as shown in Figure 11.5, “Completed Datasource Wizard”. Click the Finish button.
8. Select and double click on the Organization.ds data source. On clicking the Load Data menu in the data window the output is displayed as shown in Figure 11.6, “Organization Result”.

Note

Please refer to Appendix B, *Samples* for the sample files used in this example.

Figure 11.4. LDAP Datasource Wizard

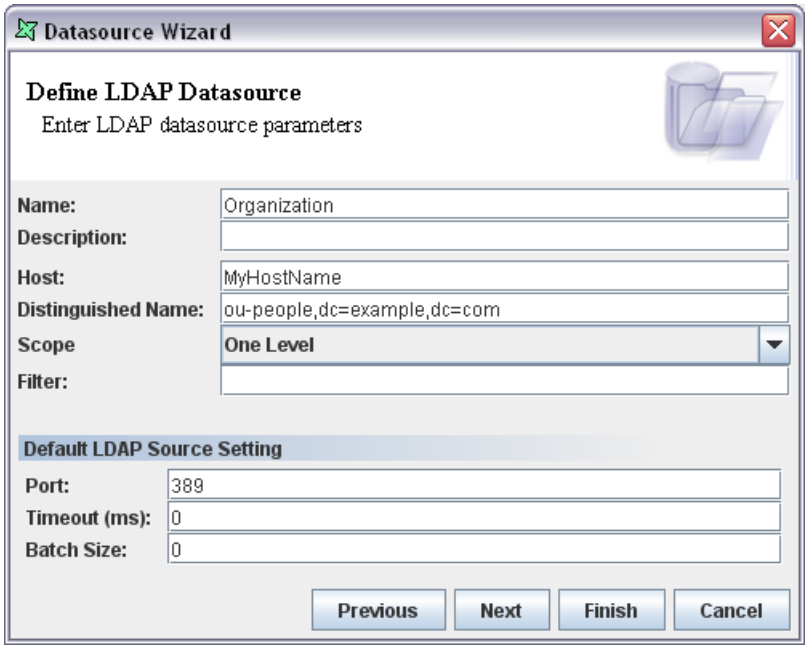


Figure 11.5. Completed Datasource Wizard

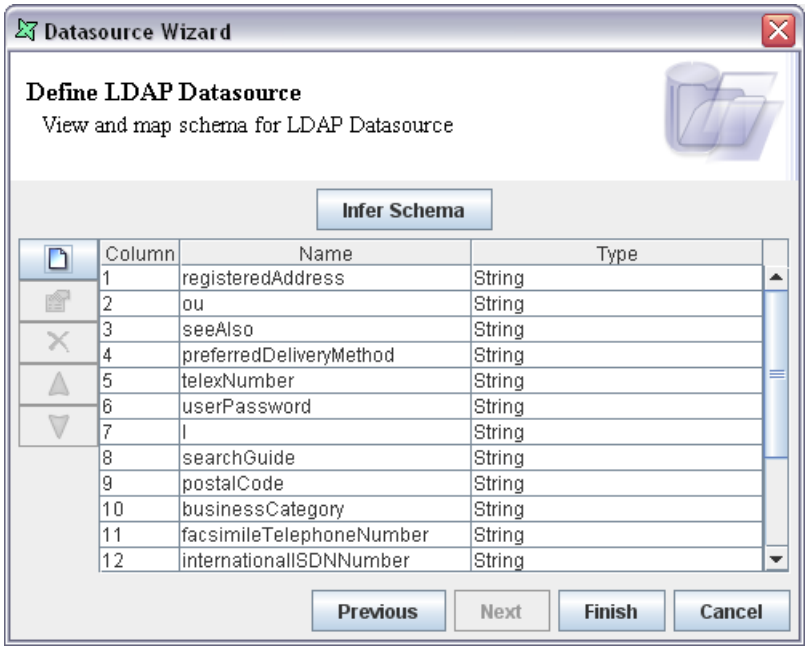
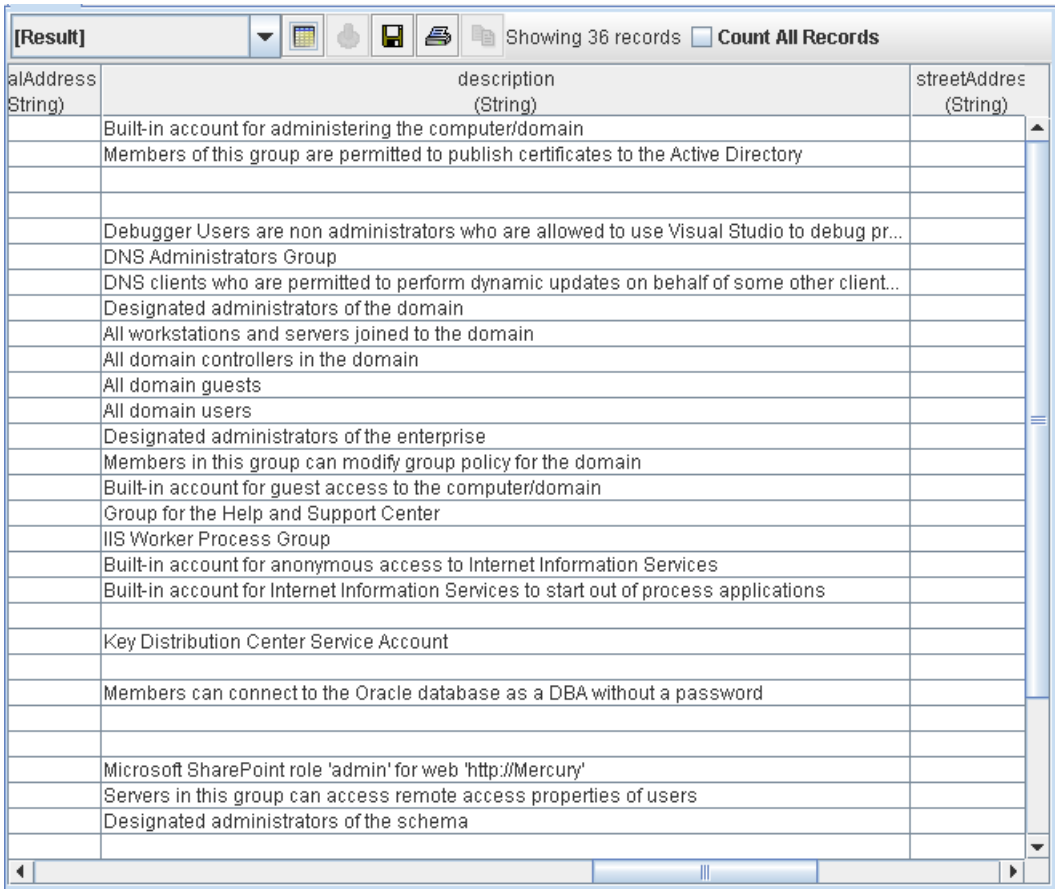


Figure 11.6. Organization Result



alAddress String)	description (String)	streetAddres (String)
	Built-in account for administering the computer/domain	
	Members of this group are permitted to publish certificates to the Active Directory	
	Debugger Users are non administrators who are allowed to use Visual Studio to debug pr...	
	DNS Administrators Group	
	DNS clients who are permitted to perform dynamic updates on behalf of some other client...	
	Designated administrators of the domain	
	All workstations and servers joined to the domain	
	All domain controllers in the domain	
	All domain guests	
	All domain users	
	Designated administrators of the enterprise	
	Members in this group can modify group policy for the domain	
	Built-in account for guest access to the computer/domain	
	Group for the Help and Support Center	
	IIS Worker Process Group	
	Built-in account for anonymous access to Internet Information Services	
	Built-in account for Internet Information Services to start out of process applications	
	Key Distribution Center Service Account	
	Members can connect to the Oracle database as a DBA without a password	
	Microsoft SharePoint role 'admin' for web 'http://Mercury'	
	Servers in this group can access remote access properties of users	
	Designated administrators of the schema	

Chapter 12
Filesystem DataSource

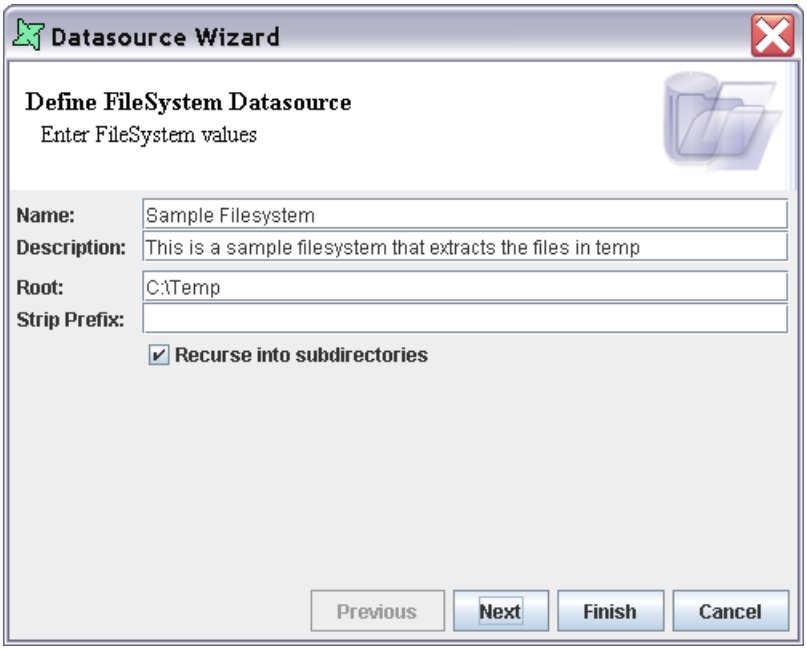
Overview

The Filesystem DataSource allows files to be processed as records. This is useful for evaluating filesystem usage, or determining which files to use as data sources. For example, an XML DataSource could use a parameterized URL to identify the file to load. This data source may be fed URLs by the Filesystem DataSource, in conjunction with the Properties processor, to iterate over all XML files in a directory and build a concatenated set of records from the set of files identified.

Filesystem DataSource Wizard

The Filesystem DataSource Wizard is shown in Figure 12.1, “Filesystem DataSource Wizard”.

Figure 12.1. Filesystem DataSource Wizard



Name: Enter the DataSource name in the text box. This should be a unique name.

Description: Any extra description that is used to describe the data source can be entered here.

Root: Choose the root directory from where files will be chosen.

Strip Prefix: This allows characters to be stripped from the beginning of the directory and path. For example the file C:\Temp\Sample.ds with Strip Prefix value C:\ will give a Directory of Temp and a Path of Temp\Sample.ds. Note that the prefix is not stripped from URLs - they remain absolute.

Recurse into subdirectories: If this checkbox is enabled, files will be selected from the root directory and any subdirectories recursively.

The records returned by this data source can be manipulated by subsequent processors. For example to obtain only XML files, you can filter by extension. To get the files sorted by name, size or date, just add the appropriate sort processor.

Filesystem Schema

The Filesystem Datasource returns records according to a fixed schema that contains the following fields:

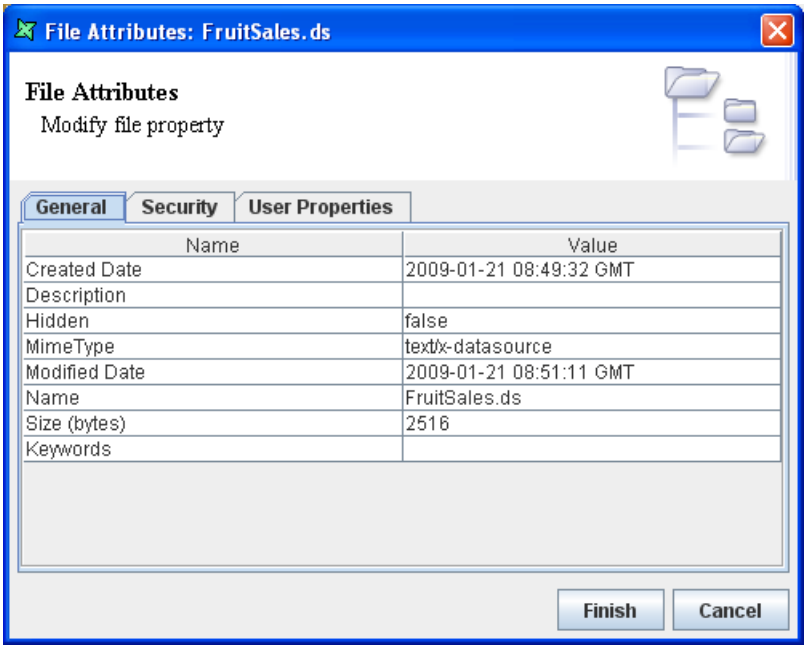
- **Name (String)**: The file name without any extension (the characters after the last '.').
- **Extension (String)**: The file extension (not including the '.').
- **Directory (String)**: The directory containing the file.
- **Size (Long)**: The file size in bytes.
- **LastModified (Long)**: The numeric value of the last modified date.
- **LastModifiedDate (Timestamp)**: The timestamp value of the last modified date.
- **Path (String)**: The full path of the file, comprising directory, name and extension.
- **URL (String)**: The url of the file, using the file:/ protocol.

FileSystem DataSource in DBFS FileSystem In Remote Designer

Here is how to use `User Properties` which is applicable to all files in a DBFS FileSystem on Remote Designer.

1. Create a DBFS filesystem using Elixir Repertoire Server web interface.
2. Add files into the filesystem by dragging them from a folder in Windows Explorer or files in other filesystems while holding the *Ctrl* key, right-click on file in other filesystem and select `Copy` or create files in filesystem explicitly.
3. Right-click on any individual file and select `Edit File Properties...`. A window which is similar to Figure 12.2, “File Attributes” will appear. User can enter some keywords for easy reference.

Figure 12.2. File Attributes



4. In the `User Properties` tab, user can add or modify user property. For demostration, we will add an additional user property with *Creator* for `Name` and *Thomas* for `Value`. Click on `Finish` to confirm all changes.

5. Next, create a FileSystem DataSource in the same DBFS filesystem. For `Root`, enter the path of the DBFS filesystem. It should be something like: *repository:/<DbfsFileSystemName>*.

6. In the next page of the wizard, user will need to enter the name of the property entered in the file previously, which was *Creator*. Note that this is case-sensitive. Click `Finish` to confirm the changes.

7. Load the datasource. By scrolling to the outmost right of the loaded data, user will be able to find two additional columns. They are `Keywords` and `Creator`. The keywords and user property entered for the files will be displayed.

Chapter 13
Tabular DataSource

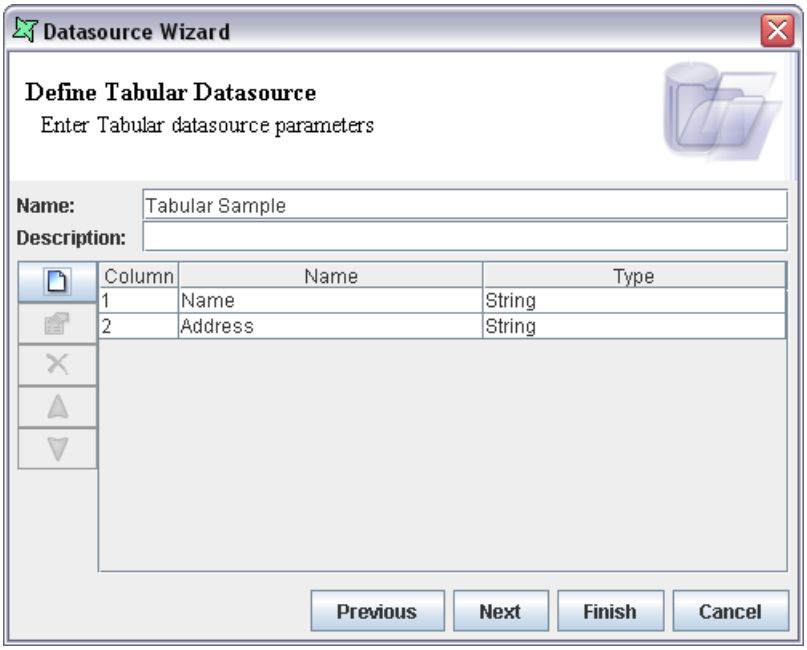
Overview

The Tabular DataSource is a self-contained data source that is editable from within the data designer. The data source contains both the data schema and the data records, in XML format, making it a very efficient mechanism for accessing small volumes of data quickly. Furthermore, the Elixir data stores are capable of generating into the tabular format, allowing any data source to be quickly replicated in tabular form. This enables users to work with data without needing a database connection.

Tabular DataSource Wizard

The Tabular DataSource Wizard is shown in Figure 13.1, “Tabular DataSource Wizard”.

Figure 13.1. Tabular DataSource Wizard



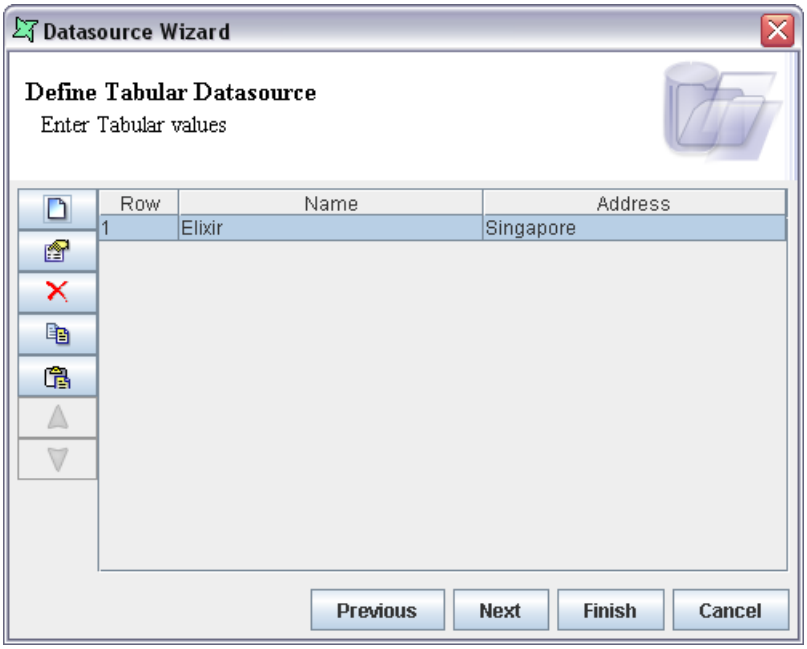
Name: Enter the DataSource name in the text box. This should be a unique name.

Description: Any extra description that is used to describe the data source can be entered in the Description text box.

The remaining UI items on this page allow you to define the schema for the records you wish to create. In the screenshot shown above, you can see two columns defined, Name and Address, both of type String.

The second page of the wizard is shown in Figure 13.2, “Tabular DataSource Wizard Page Two”.

Figure 13.2. Tabular DataSource Wizard Page Two



This page allows you to enter records that conform to the schema defined on the previous page. In the sample above, a single record, Elixir, Singapore has been added.

The Tabular DataSource also supports security options, such as read-only, hide-details and encryption. These are accessed from the third wizard page. By applying these options, you can protect the contents of the data source from being modified or even viewed outside the tool.

Note

See DataDrop and DataStore for details of how to automate the creation of Tabular DataSources.

Chapter 14
Random DataSource

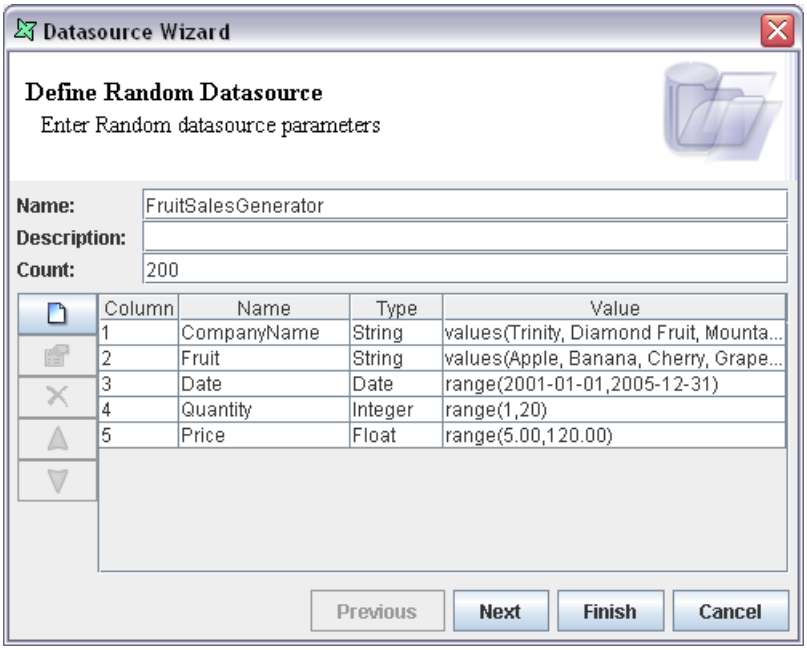
Overview

The Random DataSource is a self-contained data source that generates records that conform to a defined structure. You can choose the number of records generated and use this data source for testing large volumes of data without pre-creating all of the records. You can also connect this data source to a data store, to generate records into JDBC, Excel, CSV etc. in order to create repeatable test cases, for use in optimizing data flows and report rendering.

Random DataSource Wizard

The Random DataSource Wizard is shown in Figure 14.1, “Random DataSource Wizard”.

Figure 14.1. Random DataSource Wizard



Name: Enter the DataSource name in the text box. This should be a unique name.

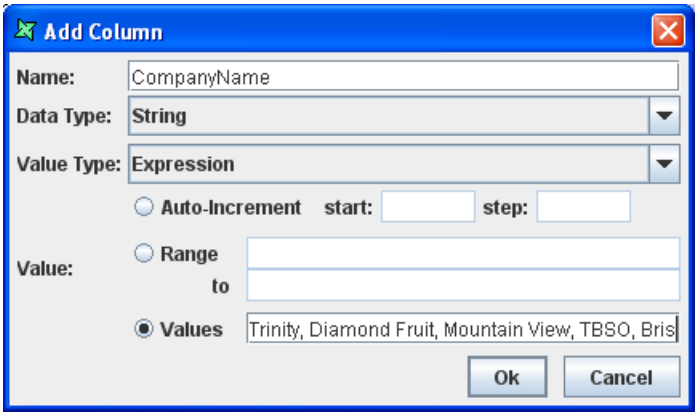
Description: Any extra description that is used to describe the data source can be entered in the Description text box.

Count: The number of records that should be generated. You can use ${substitution} of dynamic parameters here, so that the caller can supply the count value.

The remaining UI items on this page allow you to define the schema and value options for the records you wish to create. In the screenshot shown above, you can see five columns defined.

When a column is created or edited, you will see Figure 14.2, “Edit Column”.

Figure 14.2. Edit Column



This dialog allows you to specify the name and type of the column and then choose how the value should be constructed. Value Type allows you to choose Expression, Literal or Script.

Expression

The Expression option provides three modes, either an auto-increment value, a value range, or a set of predefined values.

Auto-increment

The auto-increment option will generate an incremental progression of values in any of the available data types. The first value will be the logical "zero" element - which is 0 for numeric types, false for Boolean, and 1970-01-01 for Dates (the beginning of the Unix epoch). When a limit of the data type is reached (eg. for byte, at 127) the value will be reset to the minimum supported value (so for byte, -128) and the sequence will proceed.

Range

If the range option is chosen, you should enter the first and last allowed values. These values may include ${substitution} strings. For example, for an Integer between 5 and 10, enter First=5 and Last=10. Range has special behaviour for Strings, Doubles and Floats.

String: The first and last strings are padded with spaces as necessary to make them the same length. Then, for each character, a random value is chosen between (inclusive) the corresponding character position from each string, so H and W might give P, for example, but never A or Z. Here's an example:

```
String Range S1000000A to S2999999Z
```

This allows realistic looking Singapore ICs to be generated. The first letter will always be S (S->S), the second leter will be either 1 or 2 (1->2) subsequent digits might be anything (0->9), finishing with an uppercase character (A-Z) (representing the checksum, which of course is random, so it won't usually be correct). Similarly,

```
String Range Mr Aaaaaaa to Mr Zzzzzz
```

allows random names. Of course you could separate Title as a set of Values (see later) Dr,Mr,Ms etc. for even more flexibility.

Doubles and Floats: To generate a random floating point number between 1 and 10, you can enter First=1 and Last=10, as you would expect. You will get a random number with the full precision of the data type, eg. 5.183456738923. If you want a number that represents an amount, you might want something like 5.18 instead. To obtain this, specify the number of decimal spaces in the First and Last strings: First=1.00 Last=10.00 will give you floating point numbers, but only to two decimal places. This option only works for up to four decimal places. Thus if you want a full precision decimal in the range 0.5 to 1.5, you should specify First=0.50000 Last=1.50000 (the five decimal places exceeds the limit of four, so full precision will be used. If you use 0.5 to 1.5, you will only get one decimal place of precision, as described earlier.

Values

The alternative to a range is to specify allowed values, these should be comma, separated, for example:

```
Normal, Warning, Critical
```

Each option here will have a one-third chance of being selected for each record. If you want more Normals, then just add more to the list:

```
Normal, Normal, Normal, Warning, Critical
```

Now Normal will occur with higher frequency than Warning or Critical. If you need a comma to appear in one of your values, you should escape it with a backslash. For example "Hello\, World, Goodbye" has two values, the first being "Hello, World". Once again, you can use ${substition} strings in the values.

Literal

The literal type supports ${substitution} and then sets the value into the record. This can be used when you want a specific value to occur in every record.

Script

The script type supports ${substitution} and then evaluates the script using the JavaScript engine and then inserts the value into the record. This can be used when you want a logical progression of values, for example an auto-increment of ID, or a calculated value, such as today's date. For today's date, it may be preferable to perform the calculation once and reuse the value for efficiency, as the script will be run for each record generated.

Chapter 15
ARFF DataSource

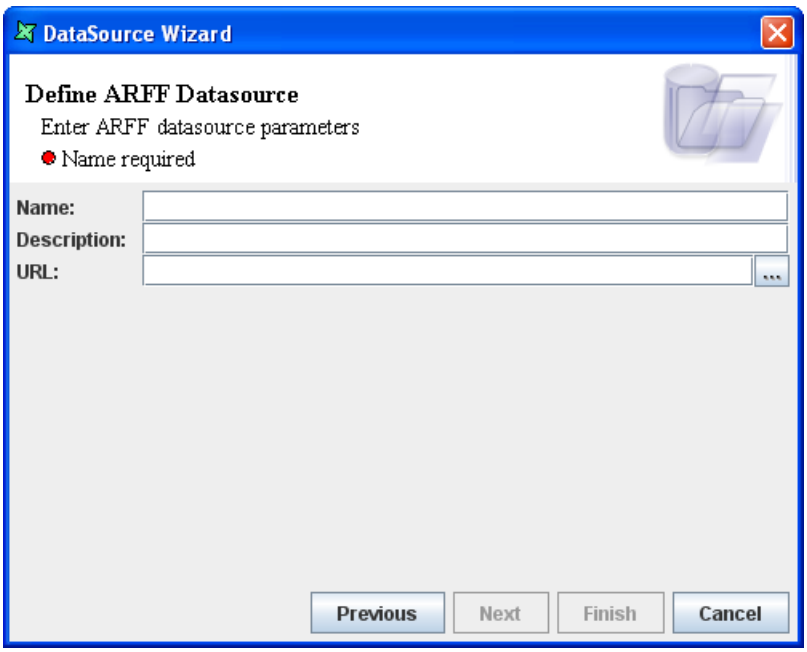
Overview

In Elixir Data Designer, the data from an ARFF file can be accessed using the ARFF DataSource.

ARFF DataSource Wizard

The ARFF DataSource Wizard is shown in Figure 15.1, “ARFF DataSource Wizard”.

Figure 15.1. ARFF DataSource Wizard



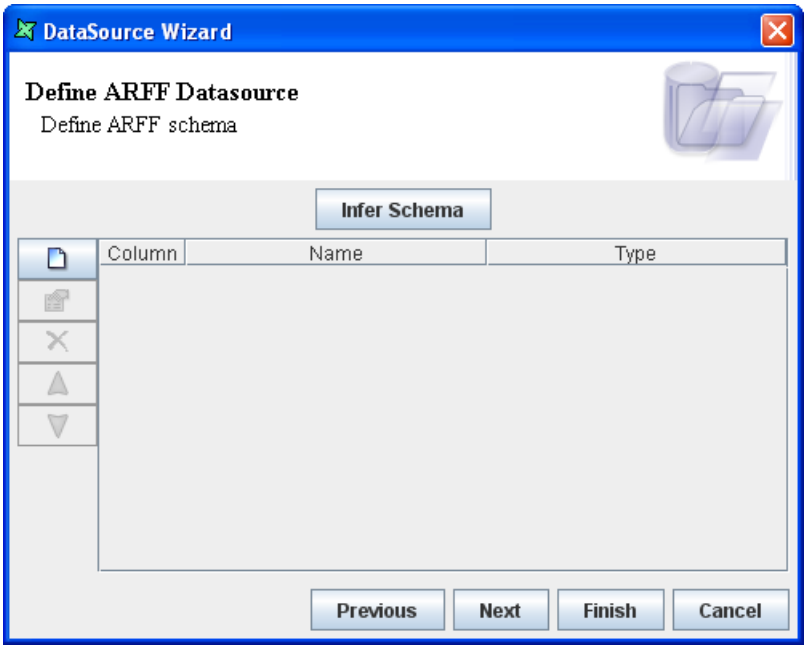
Name: Enter the DataSource name in the text box. This should be a unique name.

Description: Any extra description that is used to describe the data source can be entered in the Description text box.

URL: Enter the URL of the directory where the ARFF file is placed.

The second screen of the ARFF DataSource Wizard is shown in Figure 15.2, “Define ARFF Schema”.

Figure 15.2. Define ARFF Schema



Using the second page of the wizard, you can either enter the column names and types explicitly, or use the Infer Schema button. Where the types cannot be inferred they will default to String type. You can edit the names and types after inferring, if necessary.

Chapter 16
Cache Datasource

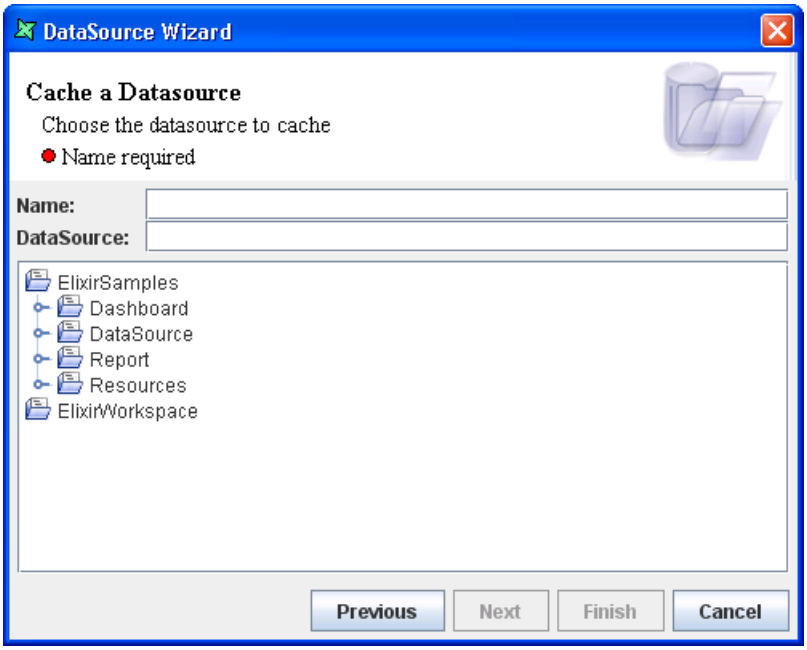
Overview

Cache datasource provides a caching mechanism to retain the data for a pre-determined time to avoid repetition of data loading and processing. The cache will remember the current set of records across multiple generations or views of the data. It is useful to use a Cache datasource when there is a long operation during flow debugging when you are likely to be repeating the same tests many times while you tune the processor logic.

Cache Datasource Wizard

The Cache DataSource Wizard is shown in Figure 16.1, “Cache DataSource Wizard”.

Figure 16.1. Cache DataSource Wizard

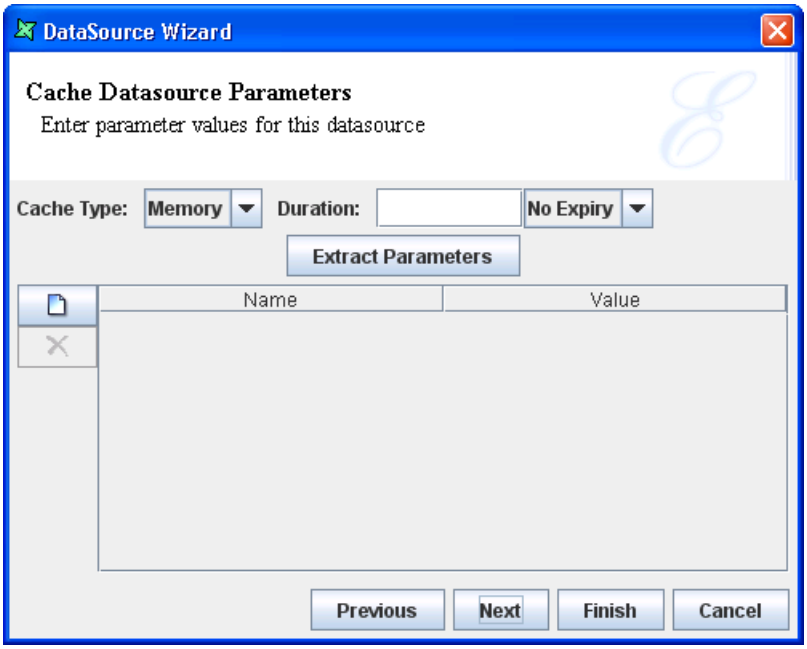


Name: Enter the DataSource name in the text box. This should be a unique name.

Datasource: The URL of the datasource used to create the Cache datasource.

The second page of the wizard is shown in Figure 16.2, “Cache DataSource Parameters”. This page allows the user to select the duration of the cache and extract the parameters from the datasource (if any). The user can also choose to enter the parameters manually.

Figure 16.2. Cache DataSource Parameters



Chapter 17
DBF Datasource

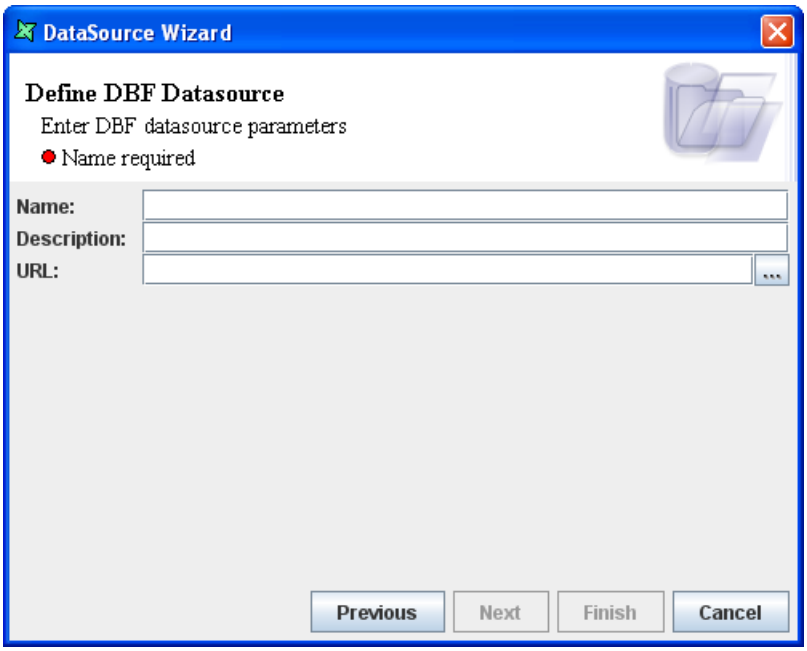
Overview

A DBF datasource is created by using a DBF file which is widely used in many applications which needs a simple format to store structured data. DBF files have been adopted in many other applications and one good example is the shapefile, which stores feature attribute data, is also used in Elixir Repertoire.

DBF Datasource Wizard

The DBF datasource wizard is shown in Figure 16.1, “Cache DataSource Wizard”

Figure 17.1. DBF DataSource Wizard



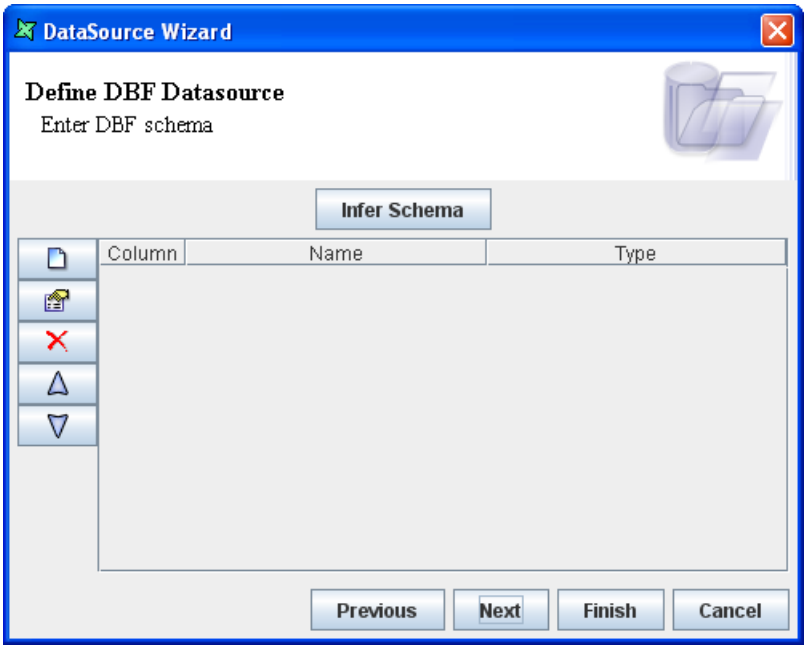
Name: Enter the a unique name for the datasource in the text box.

Description: Any additional description required to describe the data can be entered here.

URL: The URL of the .dbf file.

The next page of the wizard, seen in Figure 16.2, “Cache DataSource Parameters”, is for inferring the schema and editing it.

Figure 17.2. Enter DBF Schema



User can choose to enter the column names and data types explicitly or infer the schema using the `Infer Schema` button. Editing of column names and data types can be edited after inferring.

Appendix A. Dynamic Parameters

Dynamic Parameters

We generally need a way to control the dynamic behavior of the program by setting the values of various parameters. The dynamic parameters are specified while setting the properties of a program at design time and the value is assigned during the execution of the program.

A parameter is "declared" when you type ${something} within a text field. A field may include multiple declarations, mixed with normal text, eg. "My name is ${first} ${last}". A parameter is "defined" when you supply a value.

Dynamic Parameter Elements

Each dynamic parameter can have up to three elements, name, type and default value. These parts are separated by # characters, which can be omitted when there are no subsequent elements. Here are some examples:

- ${name} - This syntax is used to specify a parameter name.
- ${name#type} - This syntax is used to specify the name and type of the dynamic parameter
- ${name#type#value} - This syntax is used to specify the name, type and value.
- ${name##value} - If the type has to be excluded and just the default value has to be specifed the syntax is given as above.

Dynamic Parameter Names

Dynamic parameter names can include any characters, including spaces and be as descriptive as you like. For example ${Name} or ${Please enter your name}. Reserved characters such as # { and } should be escaped by prefixing with a backslash \. For example ${Enter your \#Id}.

Dynamic Parameter Types

- ## - A plain text field (the default)
- #password# - A password field so the typed characters are masked.
- #choice(A,B,C)# - A combo box with choices A, B and C.
- #date()# - A date field with a popup calendar.

  The date type provides a number of variants to control how the date is presented to the user. Within the tool, the string representation of dates always follows the international standard format yyyy-MM-dd. Here are some examples of date syntax and their meaning (the // indicates a comment and is not part of the syntax):

  - date // system locale, long format
  - date() // system locale, long format
  - date(fr) // french locale, long format
  - date(fr_CA) // french-canadian locale, long format
  - date(,"long") // system locale, long format

- date(,"medium") // system locale, medium format
- date(,"short") // system locale, short format
- date(,"(yyyy) dd/MM") // system locale, custom format
- date(fr,"short") // french locale, short format

As you can see, date can take up to two parameters (locale,format). Both parameters are optional. The locale is defined using the appropriate language, country and variant codes, separated by underscores: LA_CO_VA. Again you can omit values from the end to accept the default, LA_CO, or just LA. The language codes are defined at: http://www.loc.gov/standards/iso639-2/englangn.html, while the country codes are defined at: http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html. The format should either be "long","medium" or "short" or else a custom pattern. Remember the format only affects how the date is presented to the user, not the internal string representation of the date.

Dynamic parameters are always strings. This means the result substituted for a date type will not be a date object, only a string representation (in ISO format, as mentioned earlier). If such a string is substituted in JavaScript, you will be performing subtraction instead: 2005-01-01 = 2003! In order to convert the string to a date, you need the following form of JavaScript (showing a french date, here):

```
asDate("${Enter French date#date(fr,"long")}");
```

There are two important parts to this example. Firstly, notice that the ${} substitution is enclosed within quotes. This ensures the result is substituted as a string, not as a series of number subtractions. Secondly, the string is passed to the asDate() function, which is pre-defined within the tool for converting strings in ISO date format into actual Date objects.

- #lookup(...)# - Lookup choices from a datasource.

  It is good to give users a choice of valid values, but the options are often dynamic. Using the lookup type, you can fetch values from a datasource to populate a combo box on the parameter UI. Here's an example:

  ```
  ${Company Name#lookup(Fruit,CompanyName)#B}
  ```

  This will create a parameter on the UI with label Company Name with a combo box that reads its values from the data source called Fruit using the field called "CompanyName". The default selection will be company B.

  Further, the datasource "Fruit" can be either:

  - The name of a Graphic Object within a Composite Diagram (Data Designer)
  - The name of a Data Source within a Report (Report Designer)
  - The name of a View using view: syntax (Dashboard Designer)
  - The name of a datasource in the Repository (All)

  Also, further parameters can be passed in if the datasource requires parameters:

  ```
  ${Company Name#lookup(Fruit,CompanyName,User=Jon,Password=XXX)#B}
  ```

Ordering Dynamic Parameters

By default, parameters are presented in alphabetical order. This means, if you have parameters defined for ${Name} and ${Address} then the Address will appear first. Often we want to control the order

of presentation - it is more common to request Name before Address. To achieve this, we declare a dependency between the parameters. If the name of the parameter includes a string embedded in angle brackets <string> then this indicates that the parameter is dependent on that named parameter and should be shown after it. To ensure Address appears after Name we can therefore define ${Name} ${Address<Name>} - this declares Address depends on Name and should therefore be shown after it. Note the text in angle brackets is not shown on the UI. If we need to give type and/or default values, we include the # separator as usual: ${Address<Name>#choice(Singapore,UK)#UK}, for example.

Dynamic Parameters with a Nested DataSource

Suppose a dynamic parameter is specified in the SQL query while adding the JDBC data source to fetch the records of a specific city from the Stores table. Here's how a composite data source is added to the repository which references the Stores data source and values are passed to the dynamic parameters.

1. Make sure the Mondrian Database is available (eg. through ODBC).

2. Add a JDBC DataSource called `Stores` and choose the appropriate driver and URL for your database. If you are using Mondrian through JDBC/ODBC (as described in the section called “Using the JDBC/ODBC bridge driver”) then enter the URL as "jdbc:odbc:MondrianFoodMart".

3. Click the Next button. Enter the following SQL query in the SQL tab window.

```
SELECT DISTINCT
store.store_id, store.store_country AS Country,
store.store_state AS State, store.store_city AS City,
store.meat_sqft, store.grocery_sqft, store.frozen_sqft,
store.store_sqft FROM store Where store_city like '${Enter City}';
```

4. Click the Next button and choose Infer Schema. On the Dynamic parameter dialog, enter any city name and click the Finish button. The schema will be displayed. Click the Finish button to add the data source to the repository.

5. Now add a Composite DataSource called `Dynamic-Comp` On clicking the Finish button the Composite data source is added to the repository and opened.

6. Drag and drop the Stores.ds file onto the Composite diagram and connect it directly to the Result.

7. Open the Stores Properties from the shape on the diagram and click the Next button to see the DataSource Properties Screen. You will see the "Enter City" parameter. Enter the value as "Salem" (without the quotes) and click the Finish button.

8. Select the Result, and choose View from the popup menu. The output is displayed as shown in the fig A1.9. This is similar to that of passing dynamic parameters where Enter City=Salem;

9. Go back to the Stores DataSource Properties screen and change "Salem" to "${City1}".

10. Select the Result, and choose View from the popup menu. This time, the "Dynamic Parameters" dialog appears.

11. Enter "Salem" in the City1 text field and click on the Finish button. The Result output is the same as when the value "Salem" was hardcoded.

12. Repeat the process, replacing "${City1}" with "${City1}${City2}".

13. When you view the Result, the "Dynamic Parameters" dialog appears again, this time with two fields. Enter "Sal" in the City1 text box and "em" in the City2 text box. The text values from City1

and City2 are concatenated by the "${City1}${City2} substitution, so the records corresponding to "Salem" are fetched again.

Example Declaration of Dynamic Parameters.

The Empdata.xls consists of employee details in two worksheets which contain ranges. Using dynamic parameters we can choose during loading which range of cells to access.

1. Before adding the Excel DataSource, we need to ensure the ranges are defined in the Excel file.

   Open the Empdata.xls file and select Name -> Define under the Insert menu. The Define Name dialog box pops up. Enter name as Emp1_All and enter the range as given below in the "Refers to:" text box and click the Add button.

   ```
   =Sheet1!A$1:D$11
   ```

2. Enter name as Emp2_All and the range as given below in the "Refers to:" text box and click the Add button.

   ```
   =Sheet2!A$1:D$6
   ```

3. Enter name as Emp_All and the range as given below in the "Refers to:" text box and click the Add button.

   ```
   =Sheet1:Sheet2!A$1:D$11
   ```

   Click the Ok button in the Define Name dialog window and save the Empdata.xls file.

4. Launch Elixir Repertoire and add a new Excel DataSource called `Emp-Range` and enter the location of the Empdata.xls file as the URL.

   Select the First Row Header check box and enter the Range as given below in the text box.

   ```
   ${Range##Emp1_All}
   ```

   Click on the Next button

5. In the screen that appears click the Infer Schema button. When the Dynamic Parameters dialog pops up, enter any valid range name, from those specified above and click the Finish button. The schema will be inferred. Click the Finish button to add Emp-Range.ds to the repository.

6. After saving, the DataSource opens. On clicking the Load Data menu in the data window, the Dynamic Parameters window appears. You will notice that the text box contains the default value Emp1_All. On clicking the Finish button the data from the first worksheet is displayed in the window.

7. If instead of ${Range##Emp1_All} the parameter ${Range##} or ${Range} is entered i.e the default value is not specified. Then on clicking the Load Data menu, the Dynamic Parameter dialog with a blank text field appears.

8. Instead of the range specified in step 16 the parameter is entered as given below in the Range text box.

   ```
   ${Range#password#Emp2_All}
   ```

   Click the Finish button in the DataSource Wizard.

9. Select and double click on Emp-Range.ds. The Data window opens. On clicking the Load Data menu in the data window, you will see that the password field contains the some text (*) which correspond to the default value Emp2_All. On clicking the Finish button the data from the second worksheet is displayed in the window.

10. If instead of ${Range#password#Emp2_All} the parameter ${Range#password#} is entered i.e the default value is not specified. Then on clicking the Load Data menu the Dynamic Parameter dialog box with a blank password field appears.

11. If instead of the range specified in step 16 the parameter is entered as given below in the Range text box.

    ```
    ${Range#choice(Emp1_All,Emp2_All,Emp_All)}
    ```

    Click the Finish button.

12. Select and double click on Emp-Range.ds. The Data window opens. On clicking the Load data menu in the data window, you will see a combo box contains three values. On selecting a range value from the list and clicking the Finish button the corresponding output is displayed.

13. If instead of the range given above the parameter is entered as given below is specified.

    ```
    ${Range#choice(Emp1_All,Emp2_All,Emp_All)#Emp_All}
    ```

    Then on clicking the Load Data menu the Dynamic Parameter appears with a default value of Emp_All in the combo box. You can try changing to different values to see the different output results.

Appendix B. Samples

Availability

The Sample files for all the chapters are available in the file DataDesignerUserManual.jar which accompanies this document. If you place this file in the Elixir Repertoire ext directory before you launch the tool, the samples will be available as a read-only filesystem within the tool. You can also extract the files using a jar or zip extraction tool. For example, using jar you could type:

```
jar -xf DataDesignerUserManual.jar
```

For chapters 4 and 5, the examples are based on the MondrianFoodMart.mdb. The MondrianFoodMart.mdb can be downloaded from the Mondrian website at: http://sourceforge.net/projects/mondrian/. You only need the data files, which are available for a range of databases. Please refer to the link given below for the license.

Note

The software is subject to the terms of the Common Public License Agreement, available at the following URL: http://www.opensource.org/licenses/cpl.html.

Copyright (C) 2003-2003 Kana Software, Inc. and others. All Rights Reserved. You must accept the terms of that agreement to use this software.